COGNITIVE AND MOTIVATIONAL IMPACTS OF LEARNING GAME DESIGN ON MIDDLE SCHOOL CHILDREN

By

Mete Akcaoğlu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Educational Psychology and Educational Technology – Doctor of Philosophy

2013

ABSTRACT

COGNITIVE AND MOTIVATIONAL IMPACTS OF LEARNING GAME DESIGN ON MIDDLE SCHOOL CHILDREN

By

Mete Akcaoğlu

In today's complex and fast-evolving world, problem solving is an important skill to possess. For young children to be successful at their future careers, they need to have the *skill* and the *will* to solve complex problems that are beyond the well-defined problems that they learn to solve at schools. One promising approach to teach complex problem solving skills is using visual programming and game design software. Theoretically and anecdotally, extant research enlightened us about the cognitive and motivational potential of these software. Due to lack of empirical evidence, however, we are far from knowing if these claims are warranted. In this quasi-experimental study, I investigated the cognitive (i.e., problem solving) and motivational (i.e., interest and value) impacts of participating at the *Game Design and Learning Courses* (GDL) on middle school children (n = 49), who designed games following a curriculum based on problem solving skills. Compared to students in a control group (n =24), students who attended the GDL courses showed significantly higher gains in general and specific (i.e., system analysis and design, decision-making, troubleshooting) problem solving skills. Because the survey data seriously violated statistical assumptions underlying the analyses, I could not study the motivational impacts of the GDL courses further. Nevertheless, the GDL intervention bears implications for educators and theory.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my advisor Matthew J. Koehler for his constant support and guidance throughout my time as a Ph.D. student at MSU and during this dissertation research. I always felt lucky to rely on his good judgment before taking any step regarding my research and teaching, and I will consider him as a role model throughout my career.

I also would like to express my gratitude to Cary J. Roseth, who helped me grow as a researcher and a teacher by always taking the time providing me with guidance, and reminding me that I should "keep at it!" I will always cherish his kind support and friendship.

I also would like to thank to Carrie Heeter and Christina Schwarz for their support throughout this research. I always appreciated knowing that I can ask for help from them whenever I needed, and they would do their best to help me out.

My thanks also go to the administrators, staff and the students who agreed to be a part of the Game Design and Learning courses for the last two years. Especially, I would like to thank to Faika Topal and Canan Okatan at Ayazaga Isik Ilkogretim School; Kirk Riley and Cathy Post at ITEC, Lansing; and Gulsen Gumus and Goksel Kesim at Doga Schools.

Needless to say, my thanks also go to my dear friends and colleagues who trusted to come with me to another country and co-taught the GDL courses with me, Matthew Boyer, Kristen and Tyler DeBruler, thank you!

Finally, I would like to express by ultimate gratitude to my family and friends, who always made me feel loved. Thank you mother, father, brother, and the love of my life, my wife Asli.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

A problem can be defined as a situation where an active agent tries to reach a desired goal state from a given state by overcoming obstacles in between. The act of problem solving, therefore, is overcoming barriers to reach a certain desired goal state (Funke, 2010; Mayer, 1977). Problem solving processes involve using fundamental thinking skills and "problem solving" and "thinking" are often used interchangeably for this reason (Mayer, 1977).

Due to the importance of problem solving skills in people's daily lives, educators recognized the importance of the teaching of these skills (Resnick, 1987, 2010). Some researchers have gone so as to far to state that the teaching of problem solving skills should be the "central objective within every country's school program" (OECD, 2003, p. 154).

Despite the importance of problem solving in our daily lives, schools often fail at teaching these skills and triggering students' interest for these tasks. The heavy focus on the teaching of content knowledge is one of the reasons why schools fail at teaching complex problem solving skills. In many formal schooling contexts, this focus on content leaves students with too few opportunities to practice these vital skills that play a key role in their future careers and lives. Even when students are presented with problems, they are often simple problems, as opposed to the complex problems of the real-life. These simple problems often fall short in providing the students with engaging contexts that they can personally find worth tackling, and eventually, students are bored and do not develop interest in careers involving solving complex problems (Bennett & Monahan, 2013).

Educators and researchers acknowledged that schools do a poor job of teaching thinking skills and these skills are vital for students; hence, they have attempted to teach thinking skills

through certain school subjects that are already embedded in school curricula: for example, Latin in 1900s and technology for the last four decades. Starting in the '70s, the field of educational technology became interested in how technologies could foster children's thinking skills. Pioneering the field, Papert (1980) argued that by programming with LOGO, a simplified programming language, children could develop thinking skills. According to Papert (1980), it is during the process of programming with LOGO, students got the opportunity "to think about their own thinking, by expressing themselves in their programs and then debugging the programs until they worked" (Guzdial, 2004, p. 2). Therefore, "objects" like LOGO programming language were called "objects to think with" (Papert, 1980), and the process of tinkering has received a lot of attention for its potential to foster thinking skills (M. Resnick & Rosenbaum, 2013).

Since the '90s, a plethora of new computer languages have been developed that take advantage of new graphical user interfaces (GUI) while simultaneously connecting programming to authentic and meaningful contexts (i.e., design). These new languages have promised to enable even young children to program, by simply dragging and dropping graphical programming commands on a computer screen. Some prominent examples of new generation programming software include *Alice* ("Alice," 2012; Pausch et al., 1995), *Scratch* (Maloney et al., 1998; Resnick et al., 2009; "Scratch," 2012), *Gamestar Mechanic* (Games, 2009; "Gamestar Mechanic," 2012; Torres, 2009), *AgentSheets* (Repenning, 1993, 2012), and *Microsoft Kodu* (MacLaurin, 2011; "Microsoft Kodu," 2012). In addition to being visually attractive, these new software tools have also moved the abstract notion of programming into more concrete, inherently engaging, meaningful, and authentic contexts, such as game design, story-telling and creation of animations (Peppler & Kafai, 2009).

One new context the new generation programming software has been embedded is *game design*. Games are complex systems, made up of many interrelated variables and parameters (Fullerton, 2008; Torres, 2009). Game design is, therefore, an involved and cognitively demanding process that involves vital problem solving processes where designers are constantly challenged to make decisions, design complex systems, and troubleshoot problems as they emerge (Nelson, 2003). The digital game creation process also inherently involves knowledge of computer programming. For these reasons, game design has especially been lauded for its potential to promote children's interest in science; technology; engineering and mathematics (STEM) careers; and thinking skills (Games & Kane, 2011; Torres, 2009). Designing games, then, is also an inherently engaging task because both the creators and *consumers* enjoy the resulting product.

In both the early research with textual programming languages (i.e., LOGO), and in the last decade with the new generation software, some very important questions remain unanswered about the effectiveness of these software or programming languages to teach problem solving skills. Research findings from the 1990s are characterized by mixed results (Mayer & Wittrock, 1996), often fraught with methodological problems, and often with a focus outside of general thinking skills, such as teaching of content knowledge. New generation programming software has introduced more meaningful and engaging contexts for programming. Even less is known, however, about the effectiveness of these new-generation software for teaching of problem solving skills, because in the last decade researchers have not empirically studied the relationship between using these software and cognitive and motivational gains in students, apart from occasional reports of content knowledge gains or generic reports on motivational impacts of after-school programs (e.g., Kafai, Peppler, & Chapman, 2009). Despite providing rich narratives

to herald conceptual and theoretical possibilities of using game design software as tools to improve thinking skills and interest in STEM careers, recent research on new generation software is akin to early LOGO research in that the findings often do not go beyond providing anecdotal evidence.

Perhaps as much as leading to knowledge and cognitive skill gains, educators also desire getting students interested in or helping them see the value of a task is a very important goal for educators (Brophy, 2010), maybe as important as teaching cognitive skills. One of reasons for the importance of increased motivation is that when students' interest and value perceptions for a subject increase, it leads to increased competency perceptions and eventually success at school subjects (Hulleman, Godes, Hendricks, & Harackiewicz, 2010). It is also believed that that value perceptions are good predictors of future success and engagement in school (Hulleman et al., 2010). It is, therefore, of significance to understand specific impacts of learning game design on students' interests and value perceptions in game design, as well as in the integral components of the game design process, such as programming and problem solving.

Although early research looking at the relationship between programming or game design and problem solving has reported motivational outcomes, there are some questionable aspects that need further scrutiny. First, studies conducted in this area report changes in motivation qualitatively, or in the form of anecdotes (e.g., Harel, 1991). These qualitative reports often lack generalizability, because they come from a very small sample of individual students, and unique and personal accounts.

Second, in previous research on game design and problem solving, motivation is almost never operationalized using theories of motivation, offering instead generic accounts of student engagement and enjoyment. Such generic reports are not viable predictors for future task choice

and performance, because they are composed of multifaceted accounts of motivation (e.g., student interests, values, goal orientations). From this prior research, therefore, it is not possible to identify the specific relationships between different motivational constructs and task success, and this has little theoretical and practical value for researchers and educators alike.

Finally, we do not know if programming and game design tasks, despite being often lauded as "engaging," cultivate student interest in domains that may inherently lack trigger quality of computers or game design tasks. In other words, we do not know if game design can trigger students' interest in other domains such as problem solving and programming that are embedded in these processes. It is the collective wisdom that children enjoy working with computers (Lepper, 1985), or designing games or animations with new generation software (Utting, Maloney, & Resnick, 2010). It can be argued that some of this enjoyment is merely due to exposure to computers, because working with computers can spark temporary interest ("catch") for school subjects and motivate students (Mitchell, 1993). Although we know of the catch features of computers and game design tasks, we do not know if the interest "caught" with computers, design or programming tasks is maintained ("held") and actually changed into deeper interest and involvement in pursuing important thinking-skills, such as programming or problem solving. Through measuring students' interest and value development in game design, programming and problem solving, the change in students' motivation can be captured.

Theoretical Perspectives

This study draws on two main bodies of theory: one that conceptualizes problem-solving skills, and a second that conceptualizes motivation.

In this study, *problem solving* is viewed as the process of overcoming the obstacles that lay between the current problem state and the desired goal state (Funke & Frensch, 1995).

Success at problem solving requires possession of specific *skills*, *metacognitive functions* to call for those necessary skills at the right time and place, and the problem solver's *will*, or desire, to tackle the problem (Mayer, 1998). Problem solving is an important skill in many fields and areas such as engineering, mathematics, teaching, and medicine; namely in any domain where there are goals to reach that require cognitive effort (Funke, 2010).

Three most commonly encountered problem types that are within the scope of this study are *troubleshooting, system analysis and design*, and *decision-making*. Troubleshooting requires finding faulty parts of an otherwise operational system. It requires analyzing and understanding systems. Similarly, system analysis and design problems requires an understanding of systems, while designing systems requires one to consider multiple interrelated variables and the interplay between them all at the same time. Decision-making requires analyzing a given problematic situation, and giving the best decision by taking multiple variables (e.g., cost, time, etc.) into consideration. These problem types are encountered frequently in real-life situations and children's competence in these tasks is vital for their success in real-life as well as school settings (Jonassen, 2000; OECD, 2004a). Although these three problem types have the same underlying structure and goals, they vary slightly in terms of processes involved.

Motivational gains in this study are as important as the cognitive gains, and only when based on existing theories of motivation, they can predict future choice and performance. In this study, motivation is approached from a *perceived utility value* and *interest development* perspective (Hidi & Renninger, 2006; Hulleman, Durik, Schweigert, & Harackiewicz, 2008; Hulleman et al., 2010; Mitchell, 1993).

According to Eccles and her colleagues (Wigfield & Eccles, 2000, 2004), how much people expect to be successful at a given task (expectancy) and how much they *value* the task

(value) contributes to their future success at the task, and if they would reengage with the task. Expectancies for success and task value are believed to be positively correlated (Hulleman et al., 2010). In other words, people who believe they can successfully complete a task (expectancy for success) are more likely to be value more and persist in the task at hand, and more importantly successfully complete it, in turn develop more value and competence.

A more situation-specific value, *utility value* is believed to be a strong predictor of future interest and performance. Utility value is defined as the degree which a task is "useful or relevant for other tasks or aspects of an individual's life" (Hulleman et al., 2010, p. 881). Among the four types of task values (Wigfield & Eccles, 2000), *utility value* is believed to be specifically associated with performance and motivation, specifically with development of deeper interest and performance gains in educational settings.

Interest is defined as "the psychological state of engaging or the predisposition to reengage with particular classes of objects, events, or ideas over time" (Hidi & Renninger, 2006, p. 112). It is strongly tied with engagement decisions, performance outcomes and utility value perceptions of a task. Interest is, however, composed of multiple stages and developing deeper and more developed interest is needed for the aforementioned outcomes. Traditionally, interest is divided into two general categories: short-term or *situational interest*, and long-term or *personal interest*. According to Hidi and Renninger, interest develops in four stages, from an initial temporary interest to a fully developed personal/individual interest.

Early research indicated that through manipulating students' perceived utility value of a task, their interest for that task can be developed from early stages of interest to later stages (Hulleman et al., 2010). For example, Hulleman et al. found that even a simple 30-minute activity which helped students discover the relevance of a task to their lives had direct impact on

their interest and performance at the task. More importantly, the researchers found that this intervention was especially more effective for students with low performing or students with low expectancies for success.

Research Context

In this research, I implemented courses in Game Design and Learning (GDL) initiative to teach middle school students game design, with an explicit purpose to teach them important problem solving skills. GDL courses were also designed to lead students to have increased levels of interest and value for game design, as well as programming and problem solving tasks. These courses were offered to middle school students from various different backgrounds including students from Turkey and the US. In Chapter 3, I will describe the content and structure of the GDL courses in more detail, and, in Chapter 4, more information is provided about the student demographics and the different sites where GDL courses offered.

Purpose of the Study

The purpose of this study was to examine the cognitive and motivational changes students' show after attending GDL courses. More specifically, the purpose of this study was twofold. The first purpose was to examine the cognitive impacts of learning how to design games at the GDL courses. Specifically, capturing the changes in students' general and specific problem solving skills (system analysis and design, decision-making, troubleshooting) was of primary interest. It was hypothesized that the students attending the GDL courses would show improvements in their general and specific problem solving skills.

Using the interest development framework of Hidi and Renninger (2006), the second purpose was to examine the effects of learning game design on children's and interest development in game design, problem solving and programming tasks, as well as the changes in

their perceived utility value of these tasks. I hypothesized increases in utility value and interest perceptions in game design programming and problem solving.

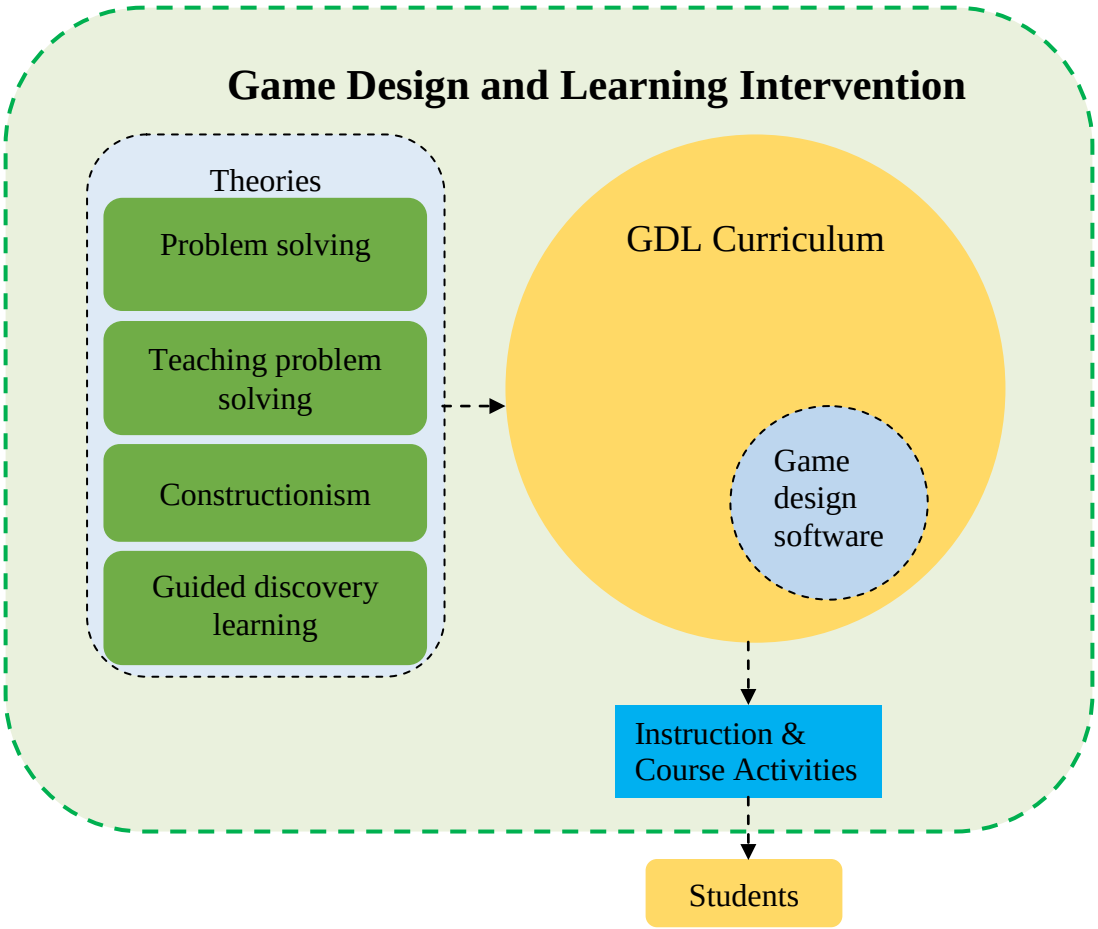
In addition, a final research goal of this study was to examine the effects of a simple intervention. The purpose of the intervention was to manipulate students' utility value perceptions of problem solving through a relevance intervention (where the students' were asked to relate problem solving skills they were learning at the camp to their school and real lives). Based on previous research by Hulleman et al. (2010), I was hypothesized that through the relevance intervention, students' perceived utility value and interest in game design, problem solving, and programming tasks would increase.

CHAPTER 2

Review of Theory and Research

The main purpose of this study was to understand the cognitive and motivational impacts of learning game design through a curriculum based on instructional methods of problem solving on middle school students. In other words, the instructional intervention (i.e. the GDL curriculum) was of central importance in this study. As it can be seen in Figure 1, however, the GDL intervention, like most educational interventions, was composed of many different layers.

Figure 1. *The Components of Game Design and Learning Intervention - For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.*



The first layer includes the theories that informed the curriculum. The main focus of the GDL intervention was to teach problem solving through game design, for this reason, the curriculum (and the instructional activities) was designed based on theories of problem solving, as well as theories regarding how to teach these skills. In addition, because during the GDL intervention students constructed socially-meaningful artifacts, the intervention was also based on Papert's constructionism (Ackermann, 2001; Papert & Harel, 1991; Papert, 1980). Finally, during the intervention, the students were guided by the instructors in minimal ways to see the connection between problem solving and game design tasks, so the intervention also was based on guided discovery learning theory.

The second layer of the intervention is the daily activities that were used during the GDL intervention and the instruction through which these activities were implemented. Specific implications of each theory on the curriculum got actualized through daily activities at the GDL courses. Inevitably, these activities were also shaped by the affordances and limitations of the game design software used (i.e. Microsoft Kodu). The nature of the instruction (i.e., the amount of lecture vs. individual activities, instructional support) was also an important part of this layer.

Finally, the last layer of the intervention reflects the characteristics of the students who benefitted from the GDL courses. The backgrounds of the students naturally play an important role in how much they benefit from such interventions. Especially in studies where data is collected from intact or self-selected groups (i.e. quasi-experimental research), these existing differences can play a significant role on the outcomes of the interventions.

The multi-faceted nature of the GDL intervention requires both an understanding of the theories that guided this research and the instructional activities, and the specifics of the day-to-day activities used during the GDL intervention. For this reason, this chapter (Chapter 2) focuses

on the underlying theories and early research that inform GDL courses and this research. In Chapter 3, I have provided a detailed description of the GDL intervention (i.e., day-to-day activities). Taken together, Chapters 2 and 3 connect the specific instructional activities and the theories while also detailing the special role of the game design software. In this research I consider the GDL intervention holistically, as looking at the impact of specific layers within the intervention was beyond the scope of this research.

Overview of Chapter 2

In this chapter, I first define what a *problem* is; the types of problems; and describe theories of *problem solving*. Then, I review effective methods of teaching problem solving skills from extant literature. I continue with a review of studies using programming or game design as an instructional method of teaching problem solving. Later in the chapter, I also explicate the theories of motivation used to frame this research and the research questions, and review previous research on motivation and learning.

Definition of Problem and Problem solving

Problem. One widely accepted definition of a problem is that "[a] problem occurs when a problem solver wants to transform a problem situation from the given state into the goal state but lacks an obvious method for accomplishing the transformation" (Mayer and Wittrock, 1996, p. 47). According to this definition, a problem is composed of a given state, a desired goal, and obstacles in between (Funke, 2010; Mayer & Wittrock, 2006; Mayer, 1977; OECD, 2012).

Problem types. Problems can differ in terms of their content, form, or involved processes (Jonassen, 2000). Out of many types of problems, in this study I picked three problem types that find frequent use in our daily lives: a) *decision-making,* b) *system analysis and design*, and c) *troubleshooting* (see Appendix1 for a detailed list of differences among these problem

types). These three problem types differ in their content, form, or process-related aspects, but these problem types also share underlying cognitive processes inherent to problem solving. Competence in solving these problems, therefore, depends on successful execution of general problem solving skills (Mayer, 1999).

Decision-making is a complex problem solving process (Funke & Frensch, 1995; Huber, 1995), involving selecting the best option from many available others (Jonassen, 2000). The complex process of decision-making involves "understanding the given information, indentifying relevant alternatives and constrains involved, constructing or applying external representations, selecting best solution from a set of given alternatives and evaluating, [and] justifying or communicating the decision" (OECD, 2003, p. 163). In decision-making tasks, the final decision can be equated with reaching the goal state of a given problem, or solving it. This process becomes difficult and complex due to the constraints in choosing each option. In real-life, we constantly face difficult situations where we need to make choices (Jonassen, 2000). For example, finding a flight that is cheap and also fast, eating a certain meal and still watching calories, and taking a certain class to finish a degree are all examples of decision-making situations that we often face. Decision-making skills are important skills to have, and success in real-life often depends on making the best decision in given situations.

System analysis and design refers to analyzing and creating systems commonly found in real-life settings. Systems are, by their nature, made up of components that are connected to each other in multiple logical ways to create unified and meaningful entities. Designing a system is a very complex task and success is dependent upon understanding the intricate relationship among its components. It involves harmony of many interrelated variables and parameters. Due to the complexity of variables involved, system design tasks are good examples of ill-defined

problems, where the goals are unclear (Jonassen, 2000; Nelson, 2003). Design problems are very common in real-life settings, and "despite their ill-structuredness, design problems may be the most important type of problem to investigate because so many professionals get paid for designing things (products, systems, etc.)" (Jonassen, 2000, p. 80). In our daily lives, to be successful we are expected to understand the systems of the groups in which we live or belong. For example, to be successful, a writer needs to understand what the systematic structure of writing a good story involves; or a gamer needs to understand the patterns of "enemies" in order to beat the game. Depending on the domain, there may be multiple systems that work, and competence in seeing patterns in these systems is often what makes individuals more successful in their lives or careers.

Troubleshooting is perhaps the most common problem solving task people face in their daily lives (Jonassen, 2000). A fact of modern life is encountering technological devices, such as "smart phones" or computers that do not work: people are often faced with situations where they have to understand how the machines work, find the inoperable part(s), and then devise and execute a solution. Troubleshooting requires problem solvers "to comprehend 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 its originally designed working state (OECD, 2003, p. 168). The essence of successful troubleshooting is understanding how a designed system works (National Research Council, 1999).

All of these types of problems share an important feature: they are each very common in real-life and are complex problems. Success in these problems plays an essential role in young children's future success in their lives and careers. Unfortunately, formal schooling does not give students many chances to practice these vital skills.

The problem-solving process. From a cognitive perspective, the problem-solving process involves successful execution of specific component cognitive processes: understanding, representing, planning/monitoring, and executing (Mayer & Wittrock, 2006; Mayer, 1977; OECD, 2003, 2012; Polya, 1957). According to this perspective, in order to successfully solve a problem, one needs to understand the problem first. *Understanding* involves using the background knowledge and making sense of the concepts of a given problem. After understanding, the next step is representing the problem. *Representing* refers transforming external representations of a problem into internal mental representations (Jonassen, 2000; Mayer & Wittrock, 1996, 2006; OECD, 2003, 2012). This process involves generating hypotheses based on interrelationships among variables of a given problem situation. After understanding the problem and creating a mental representation of it, the final step is to *plan* a solution for the problem by breaking it down its parts, and then *executing* solution (Jonassen, 2000; Mayer & Wittrock, 1996, 2006; Polya, 1957). This process involves metacognitive skills (Mayer & Wittrock, 1996, 2006) or problem solver's awareness to know when and how to execute these cognitive components successfully.

During problem solving, it is widely agreed that three cognitive components need to work together: *skill, metaskill,* and *will* (Jonassen, 2000; Mayer, 1998; OECD, 2012). These components lead to success in problem-solving tasks only when they are all present (Mayer, 1998). *Skill* refers to basic cognitive skills used to solve a particular problem. Knowing how to add, for example, can be an important cognitive skill to solve a simple addition problem. Skill is a very basic component needed for problem solving, but it cannot solely be enough for transfer of the problem-solving competency to novel contexts, as the mastery of a specific skills is often context-bound (Jonassen, 2000). *Metaskill,* or metacognitive skill, means "knowledge of when to

use, how to coordinate, and how to monitor various skills in problem solving" (Mayer, 1998, p. 53). The knowledge of how to add numbers is a simple cognitive skill, but one cannot be successful in solving a complex problem that involves addition, unless s/he knows that addition is needed in a specific case and puts that skill into practice.

Skill and metaskill are important components for successful problem solving, but without *will,* they, too, are not enough. Although motivation has not always been an integral part of problem solving theories; according to Mayer (1998), it is important for someone to successfully solve a problem, a) to be interested in the problem (i.e., *will*), b) have the belief that they have the ability to solve the problem and attribute their success to effort rather than ability (Dweck, 1999; Weiner, 1985), and c) see solving the problem as an endeavor worth dealing with (Jonassen, 2000). The presence of will, together with skill and meta-skill, makes successful problem solving possible.

Based on these perspectives, the GDL intervention involved connecting the metaskills and skills problem solving requires and the game design process. These activities will be explained in more detail in Chapter 3.

Teaching Problem Solving

There are four main empirically-proven methods of teaching problem solving skills: a) teaching *basic skills,* b) teaching for *understanding,* c) teaching by *analogy,* and d) teaching skills *directly* (Mayer & Wittrock, 1996, 2006). Although these methods are effective ways of teaching problem solving, they have varying degrees of success depending on the teaching context, so this list should not be taken as a prescriptive one. During GDL courses, activities were structured around these methods of teaching problem solving to varying degrees.

Teaching basic skills. This method encompasses teaching of specific low-level component cognitive skills pertaining to a task, so that in the ensuing tasks more effort can be placed upon executing higher-level cognitive skills (i.e., automaticity, cognitive load-reduction). According to Mayer and Wittrock (1996), while teaching basic skills, the focus of instruction should be on teaching the skills that will be required most in the later tasks. Early research showed that successful acquisition of basic skills can help in the process of solving more complex problems subsequently (Mayer & Wittrock, 1996).

Built on cognitive theories of learning, *load-reduction methods* removes the constraints in the environment that make the process of selecting, organizing and integrating of new knowledge difficult (the components of information-processing theory of meaningful learning) and thus eases the process of problem solving. Often, the lack of basic skills, or the lack of automation of them, is the biggest constraint for novice problem solvers. This lack prevents them from reaching quick and successful solutions. Removing those constraints significantly improves the problem-solving process. For example, giving calculators to students during a math test can reduce the "computational constraints" (Mayer & Wittrock, 2006). Methods that help reduce cognitive load by either automating the necessary cognitive skills or outsourcing them are effective methods of teaching problem solving.

Empirical support for teaching of basic skills on subsequent higher-level tasks comes from Glynn, Britton, Muth, and Dogan (1982). In their study, the authors investigated the impacts of whether removing structural constraints (e.g., mechanics) from students' writing tasks led them to write more persuasive arguments in their writing tasks. The results of the study showed that when such mechanical constraints are removed, students write better persuasive arguments. To this end, the authors argued that "when content and structure operations are

demanded simultaneously from writers, processing capacities are taxed severely" (p. 565). As this research shows, when low-level cognitive tasks are removed (or automatized), problem solvers have better chances to execute higher-level cognitive skills and can solve more complex problems.

During the GDL intervention (see Chapter 3), the curriculum was purposefully designed to make sure that the students master the usage of software, basic programming skills, and game design before they were given scenarios of problems and asked to replicate them within the software.

Teaching for understanding. In teaching for understanding, teaching of new cognitive skills happens in such a way that the freshly learned skills can be applied to novel situations (Mayer & Wittrock, 1996). This requires providing learners with opportunities to relate the existing and new knowledge during the learning process. Learners, in this process, are encouraged to actively construct their own learning outcomes (Bruner, 1961; Mayer & Wittrock, 1996), and through this construction process they form meaningful links between the new and existing knowledge.

Structure-based methods are one way to ensure that teaching for understanding occurs. In structure-based methods, learners are given concrete objects that can be manipulated. Through the process of manipulation, the learner understands the underlying structure common to both the object and the problem at hand. Using computer programming to teach problem-solving skills can be considered as an example of structure-based methods. Similarly, the activities in the GDL intervention made use of a game design context and software to give students a hands-on experience in solving complex problems, and can be considered as an example of structure-based methods.

Empirical support for the effectiveness of structure-based methods in teaching of problem solving skills comes from a study by Moreno and Mayer (1999). In their study, the researchers conducted a series of experiments to understand the role of concrete models in understanding abstract mathematical concepts. In Experiment 1, the researchers tested the effectiveness of two different instructional methods while teaching a simple math concept (adding and subtracting using signed integers) to sixth graders (n = 60). The control group received single-representation (SR) of the arithmetic procedure through only symbols (i.e., numbers). The experimental group, on the other hand, received instruction through multiple-representations (MR), symbolic, verbal and visual form. The results of the study showed that for difficult problems, the MR group did better than the SR group, while for easy problems, due to ceiling effect, both groups scored similarly. In Experiment 2, to test the interaction between cognitive-load theory (low-spatial ability students cannot make use of visual aids due to reaching their cognitive limits sooner) and MR methods, the authors looked at the performance difference between high and low-spatial ability students. The results showed that high-spatial ability students produced a significantly greater pre-test-to-post test gain than did the low-spatial ability students, confirming that cognitive-load theory holds true and MR methods are most effective with high-spatial ability, or higher-achieving students.

In addition to structure-based methods, *generative methods* are also effective in promoting teaching for understanding by helping students "generate relations between their existing knowledge and information to be learned" (Mayer & Wittrock, 2006, p. 294). Prominent examples of generative methods include *elaborative methods*, where the learners are asked explicitly to make the connection between old and new material; *note-taking methods* where the learners are asked to summarize new information, and make connections with existing

knowledge; *self-explanation methods* where the learners are asked to explain concepts while learning; and *questioning methods*, where the problem solver is encouraged to ask questions about the things they are learning. During the GDL intervention, students spent time analyzing problem scenarios and recreating (i.e., generating) them within the game design software. Such a process of reverse-engineering a problem scenario can be considered as an example of using generative methods to teach problem solving.

In an early study by Linden and Wittrock (1981), the authors provided empirical evidence for the effect of generative methods in fostering reading skills in children. According to the generative teaching method designed by the researchers, they theorized that the students' comprehension in reading could be boosted by "inducing the readers to attend to the text, to relate their knowledge and experience to it, and to build associations, abstractions, and inferences from it" (p. 45). To test their hypotheses, the researchers conducted an experimental study. The participants were 64 fifth-grade students and they were assigned to one of the following conditions randomly: 1) Imaginal to Verbal Generations, 2) Verbal to Imaginal Generations, 3) No Instructions to Generate, and 4) Classroom Teacher Taught Control Group. After the treatment session, the researchers measured the students' factual knowledge of the texts and their comprehension of them. The results of the study showed that the groups who received "generation treatment" scored significantly better at fact retention and comprehension tests. There was a positive correlation between generations and comprehension. This study clearly shows the effect of generative activities in a context-specific problem-solving task, reading.

Finally, using *discovery methods* is another instructional approach that promotes teaching of problem solving for understanding. During the discovery learning process, it is believed that learners actively construct connections between existing and new information, which eventually

leads to meaningful learning. Nonetheless, one distinction should be made between pure and guided discovery methods. In pure discovery methods, learners are left on their own to make the connections, while in guided discovery learners are coached through the process to make sure that they establish the necessary connections. Guided discovery methods are known to be superior to pure discovery methods (for a review see: Kirschner et al., 2006; Mayer, 2004). During the GDL intervention, as it will be explained in more detail in Chapter 3, students were placed into a guided discovery setting: instructors provided guidance to get students started with game design and problem-solving tasks, and provided them with chances to explore solutions through discovery.

In a recent study, Moreno (2004) provided empirical evidence for the superiority of guided discovery methods over pure discovery methods. In a series of experimental studies, learning software was tweaked to create two versions, one which provided guidance (Explanatory Feedback [EF]) and another which only provided "Corrective Feedback" (CF) and no guidance. During experiment 1, a group of college students (n = 49) learned how to "design a plant" using the two different versions of a learning software. Explanatory feedback in the experimental group was provided in the form of guidance during the selection of plant parts. When the students received EF, it explained why their choices of a certain part were suitable or not. The students' retention of botany knowledge was the main measure of the study, while transfer, motivation, interest, helpfulness, and difficulty of using the software were also measured. Statistically significant differences between the EF and CF groups were found for all measures. Specifically, groups that received EF retained significantly more knowledge than the CF group. EF group also did significantly better in the transfer test.

Teaching by analogy. In this approach, "learners solve a new problem by using what they know about a related problem that they can [already] solve" (Mayer & Wittrock, 1996, p. 55). In teaching by analogy, the premise is that by analyzing two problems that have surface similarity, the learners can abstract the underlying rules. Understanding the *base* problem, learners are asked to generate a solution for the *target* problem that is similar to the base problem. In empirical studies, researchers found that when not guided, the learners often fail to make the connection between the base and target problems, even though the problems are fairly similar (Mayer & Wittrock, 1996). Teaching by analogy was implemented throughout the GDL intervention in the format of providing structurally similar scenarios to students and asking them to explore the patterns among different cases. One instance of guidance was, for example, the instructor's going through the problem-solving steps together with the students, showing them what specific cognitive skills they need to adopt for each specific problem.

Gick and Holyoak (1983, 1980) conducted multiple experimental studies to understand the impact of using analogies while teaching problem solving. The results of their comprehensive set of studies indicated that subjects needed hints to be able to establish analogies between analogous problems (1980). They also identified in another set of studies that (1983) people positively benefit from seeing a pair of analogous problems first, identifying the analogy between them, and then working on a third problem. The authors also argue that when an analogy between two problems is established, any visual or verbal mechanisms that support the analogy can significantly enhance the problem solving on the subsequent analgous problems.

Teaching thinking skills directly. Another method of teaching problem solving is teaching them directly (Mayer & Wittrock, 1996, 2006). By teaching problem-solving skills directly through after-school programs or courses, early research showed that it could be an

effective way of teaching problem solving. Based on the fact that transfer of skills is more likely to occur within similar domains, it is argued that *teaching domain-specific thinking skills* is often more successful than teaching domain-general skills. Within the GDL intervention, activities also centered on showing students the importance of using basic skills to solve complex problems. For example, students were taught the use of the problem solving steps by Polya (1957): understand the problem, devise a plan, carry out the plan, and evaluate.

In a recent study, Quilici and Mayer (2002) provided empirical evidence for the effectiveness of teaching a domain-specific problem solving skill in solving problems. Specifically, the researchers investigated the effectiveness of directly teaching "structural awareness" in improving college students' statistical problem solving skills. The study incorporated a within-subjects design, where a group of undergraduate students (n = 51) took a problem-sorting test at the beginning and end of an introductory level statistics class. During the class, the students learned about *t*-tests, chi-square analyses, and correlation; and these were the topics covered in the pre and post-tests. The results showed that after learning the domain-specific skill of analyzing statistical problems, the students were significantly better at recognizing and sorting problems. Second, the students made use of word structures (a skill they specifically learned) while sorting problems in the post-test significantly more than they did in the pre-test. In a follow-up study, the researchers tested if even a short-term aid, instead of a full semester course, could foster structural awareness in statistics word problems. The results showed that the students who received example aids did significantly better in sorting problems based on their structural features rather than their surface features, and it was seen that the experimental groups did significantly better than the no aid group. The results of these studies

show that it is possible to teach domain-specific skills for an improvement in problem solving skills in that specific domain.

Research on Using Computers to Teach Problem Solving

Using programming. With the increasing popularity of LOGO in 1980s, an interest in the use of programming as a means to teach problem solving skills started. Researchers (Guzdial, 2004; Papert, 1980) argued that both the programming process and the embedded debugging process were important factors that contributed to peoples' thinking skills. Despite this focus on establishing the connection between problem-solving skills and programming, however, most research did not provide detailed accounts of their teaching interventions. We are far from knowing what theories they were based on and if they made use of the theories of teaching problem solving (e.g., teaching by analogy, structure-based methods, etc.).

Choi and Repman (1993) conducted a study to find the difference between two programming languages (Pascal or FORTRAN) on students' problem-solving skills. Data was collected from 58 undergraduate students taking one of the three classes: programming with Pascal (n = 18), FORTRAN (n = 19), and basic computer literacy (n = 21). In order to measure the change in students' thinking skills, the subjects were given a 61-item cognitive skills test before and after the semester. It was found that there were significant differences between the control and experimental groups, and there was not a significant difference between Pascal and FORTRAN groups. It was concluded that learning to program with Pascal or FORTRAN improved students' problem-solving skills. Methods of teaching problem-solving skills, however, were not reported in this study.

One of the rare studies reporting on an open connection between problem-solving methods and programming is one that created a curriculum with an explicit focus on connecting

these tasks and was conducted by Palumbo (1990). In his study, Palumbo investigated the relationship between programming and problem solving. The data came from two groups of high school students: programming group (n = 11) and basic computer literacy group (n = 11). In order to measure the change in the students' problem solving skills, a 61-item test was given to the students before and after they completed their classes. The results showed that there was a main effect of the treatment: the students in the Basic programming group outperformed the control group on the post-test. The students' computer anxiety was also measured. Both groups showed a decrease in their anxiety of computers, compared to before taking the computer classes. For the programming group, there was also a positive relationship between computer performance and problem solving. Being strictly grounded in problem-solving theory, the curriculum used in this study provided a link between problem solving and programming tasks. Nonetheless, the details of the curriculum were not reported in this research, so it is not possible to identify the exact impact of the instructional methods on teaching problem solving.

In a follow-up study, in order to understand the effects of two computer-based environments, Palumbo and Palumbo (1993) compared problem solving skills of a group of fifth-grade students (n = 15) who were taking a computer literacy class to another class of students (n = 15) who were learning programming with LEGO. Using a 51-item problem-solving test, the researchers collected data on the change in students' problem-solving skills. The results showed that, although there was not a significant difference between the LEGO group and the control group, there was a significant change in the students' problem-solving skills in both groups. Methods of teaching problem solving, however, were not reported in this study.

Nastasi, Clements, and Battista (1990) conducted a study to see the effect of LOGO instruction on the students' (n = 40) mathematical problem-solving ability. The control group

students attended a computer class where they learned how to use various software without any specific focus on problem solving. A paper-and-pencil problem-solving test (test of executive level problem-solving processes) was given before and after the treatment to capture the change in students' problem-solving skills. The results of the study showed the LOGO group did significantly better than the computer-assisted instruction group on the problem-solving post-test. This study provides support for the relationship between learning to program and thinking skills. Methods of teaching problem solving skills, however, were not reported in this study.

In another study, Gallini (1987) compared the effects of learning to program with LOGO with a regular computer literacy class in fostering problem-solving skills. Forty-four fourth-grade students were randomly assigned to one of the two conditions. While the LOGO group learned how to program using LOGO, the computer literacy group also received instruction on flowcharting and fundamentals of programming with BASIC. A ten-item test measuring the students' abilities in "following the directions" and "formulating directions" were given to the subjects before and after the treatment. The authors found that LOGO group did better in the problem-solving test than the students who attended a traditional computer literacy class. In addition, it was seen that both groups showed improvement from pre to post test. It was concluded that there was a positive relationship between learning to program in LOGO and problem-solving skills. Methods of teaching problem-solving skills were not reported in this study.

Finally, in two meta-analysis studies (Liao & Bright, 1991; Liao, 2000), it was seen that the students taking computer programming classes outperformed the students who did not learn programming, as measured by the cognitive skills tests used in the studies. In the first meta-analysis, Liao and Bright (1991) collected effect sizes from 65 studies. The overall grand mean

of the study-weighted effect size was 0.41. This implies that students who learned how to program in these studies outperformed the students who did. In the later and more recent meta-analysis study, Liao (2000) compared data from 22 studies conducted between 1989 to 1999. The overall grand mean of the study-weighted effect sizes were 0.76, indicating that the students who learned programming in these studies showed greater performance improvements than the students who did not. As a result of these two meta-analysis studies, Liao (2000) argued that "the outcomes of learning a computer language extend beyond the content of that specific computer language. Students are able to acquire some cognitive skills such as reasoning skills, logical thinking and planning skills, and general problem solving skills through computer programming activities" (p. 568).

Using game design. Despite their potential to teach thinking skills and the popularity of game and software design contexts for after-school or summer programs, empirical support showing the impact of these activities on problem-solving has been lacking. Review of research shows that there are not any empirical studies looking at the relationship between game design and problem solving, while a review of research on software design mainly shows reports of gains in content knowledge. In addition, similar to the research on programming and thinking skills, researchers in this area also did not provide detailed accounts of their interventions, or curricula, therefore it is hard to identify if they made use of theories of teaching problem solving.

In order to understand the effectiveness of creating multimedia software on thinking skills and science knowledge, Kafai, Carter Ching, and Marshall (1997) conducted a quasi-experimental study. During the study, using LOGO, 26 fifth and sixth grade students worked on creating interactive multimedia software that would be used to teach about astronomy for younger students. The students spent 46 hours on the project, twenty-three of which was

dedicated to programming activities. Data on the students' knowledge of astronomy and LOGO programming was collected before and after the camp. The results showed that there was a significant improvement in students' understanding of astronomy. Similarly, students' LOGO knowledge also improved. This study provides empirical evidence for knowledge improvement through design activities. It does not, however, provide evidence for students' improvement in thinking and problem-solving skills. In addition, we do not know if there were also changes in students' motivation to learn science or continue working on programming tasks in the future. Methods of teaching problem-solving skills were not reported in this study.

In an earlier quasi-experimental study, Harel (1991) investigated the effectiveness of creating software with specific purposes (i.e., teaching fractions) using LOGO on children's basic concept of fractions. To test her hypothesis, 51 students were placed into three classrooms: one experimental, and two control groups. The experimental group was composed of 17 fourth-grade students. The students in the experimental group followed Instructional Software Design Project (ISDP) curriculum, which integrated software design activities, LOGO programming, and math concepts (i.e., fractions). Neither of the control groups had math embedded into their curriculum, and learned fractions through regular math classes. The experimental group, in addition to their regular math classes, also had a chance to put their math knowledge into practice by creating interactive software that teaches young children about fractions using LOGO. The change in students' knowledge of fractions was measured by a 65-item multiple-choice pre- and post-test. There was a statistically significant difference between the groups on the post-test, and the experimental group significantly outperformed the other two groups. These results provided evidence for the importance of meaningful design projects using LOGO, as opposed to "just" learning how-to use LOGO. In an embedded curriculum, learning with LOGO

was effective in improving children's math knowledge. This study was one of the first to provide empirical evidence for the potential of LOGO. It, however, did not provide evidence for the change in children's motivation for math, or the changes in children's thinking skills. In this study, the author did not make any specific references to any theories of teaching problem solving, and we do not know how their intervention was structured.

In a recent qualitative study, Richards and Wu (2011; Wu & Richards, 2011) explored whether game design activities were effective in teaching computational thinking skills, using data from several sources (reflective short answer questionnaires, field notes, student game design journals, and semi-structured interviews). Apart from providing a little anecdotal evidence, this research, however, did not provide any empirical results. Despite the lack of evidence, however, the authors argued that at the end of the research the children learned how to design games and improved in their understanding and application of computational thinking skills. Methods of teaching problem-solving skills were not reported in this study. Therefore, we do not know if their intervention and activities were based on any theories of teaching problem solving.

Games and Kane (2011) also looked at the relationship between game design activities and the students thinking skills and STEM knowledge. The data for their research came from 36 middle-school students who were attending *Science and Art of Game Design* (SAGD) and Globaloria after-school camps. During these camps, the students learned how to design games using Microsoft Kodu, Gamestar Mechanic, and Adobe Flash. Based on interview data, observations, and the games the students created, the researchers tried to find evidence of improvement in students' thinking skills and STEM knowledge. No statistical analyses were provided in this research, and only anecdotes from some students were provided as evidences of

improvement. Based on the evidence, the researchers argued that SAGD and Globaloria classes provided learning environments that help youth develop "habits of mind and practice that are fundamental to the STEM disciplines" (p. 7). Although from anecdotal evidence we see that there were instances of children learning how to design games, this research did not provide any evidence for change in thinking skills and motivational changes in students. Similar to other research in this domain, there were not any references to any connection to theories of teaching problem solving in this study, and we cannot know the nature of their intervention.

Summary. Early research with simple textual programming languages found promising evidence for the impact of using programming over simple computer-assisted teaching sessions. Learning game design has been a popular method of teaching thinking skills (i.e., computational thinking), but recent research on game design have given us little reason to believe that there is, in fact, an empirical connection. Although there is some anecdotal evidence pointing to a potential connection between game design and thinking skills, it is hard to know if learning to design games using the new generation visual programming software causes improvements in thinking skills. Especially, the connection between game design and problem solving skills needs further scrutiny.

Motivational Constructs in Learning

Motivation can be defined as a theoretical construct to explain the initiation, direction, intensity, persistence, and quality of behavior (Brophy, 2010; Maehr & Meyer, 1997). It is believed that a person's choice, persistence, and performance in a given activity is affected by certain motivational constructs (Wigfield & Eccles, 2000, 2004). It is, therefore, important to measure changes in motivation to predict future behavior.

Expectancy-value theory. One theory of motivation that is a strong predictor of students' future preferences and success in a given domain is *expectancy-value theory* (Wigfield & Eccles, 2000, 2004; Wigfield, 1994). According to expectancy-value theory, students' self-efficacies and value perceptions of school subjects are important predictors for their success in these subjects in the future (Wigfield & Eccles, 2004). Expectancies and values directly influence *choice*, *performances*, and *persistence* on a given task (Brophy, 2010; Wigfield, Eccles, Roeser, & Schiefele, 2008; Wigfield & Eccles, 2000).

According to this theory, our willingness to start or the effort we put into an activity depends on how much we value the activity (or the outcomes), and how much we expect to be able to complete it successfully (Brophy, 2008, 2010; Wigfield et al., 2008; Wigfield & Eccles, 2000). In expectancy-value theory, effort we put into an activity is viewed "as the product rather than the sum of the expectancy and value factors because it is assumed that no effort at all will be invested if either factor is missing entirely" (Brophy, 2010, p. 16). In other words, if people believe that they lack the necessary skills or do not see the activity to be an important task, then the effort they will put into completing this task equals to zero.

Two important components of the expectancy-value theory, as the name suggest, are self-efficacy perceptions and value attributions. Self-efficacy can be defined as "judgments of how well one can execute courses of action required to deal with prospective situations" (Bandura, 1982, p. 122). Some researchers believe that self-efficacy alone can predict the amount of effort people will expend and if people will persist in the face of difficulties at a given task (Bandura, 1982; Pajares, 1996).Value aspect of motivation explains how much worth people see in a given task (Brophy, 2010). Despite having been understudied (Brophy, 1999, 2008), according to Brophy (2010) and expectancy-value theorists, value is equally important in predicting effort.

Depending on a combination of the self-efficacy and value attributions on a given task, the effort someone puts into a task changes significantly (see Table 1). For example, students choose to be *engaged* in a given task when both value and self-efficacy is present, *dissembled* when the value is seen but self-efficacy is missing, *evade* when have self-efficacy but do not value the task, and finally, *reject* the task when both success and value are missing (Brophy, 2010; Hansen, 1989).

Table 1. *Student Behavior Based on Expectancy and Value Perspectives - Based on Brophy (2010) and Hansen (1989)*

	- efficacy	+ efficacy
- value	Rejection: Does not Participate	Evading: Does the Minimum
+ value	Dissembling: Protects image of Competence	Engagement: Seeks to learn

Task values. Values in expectancy-value theory are more situation-specific, and, according to Wigfield and Eccles (2000), task value encompasses four different value types: attainment value, intrinsic value, utility value, and cost. Although researchers have studied them together and separately in different contexts, perceived utility value, or the extent that a task is seen as relevant one's life or other tasks, has been seen as an important factor impacting future performance and interest in school subjects (Hulleman & Harackiewicz, 2009). Recent research have shown that increased utility value predicts increases in interest and performance, especially for students with low self-efficacy or performance (Hulleman et al., 2010; Hulleman & Harackiewicz, 2009).

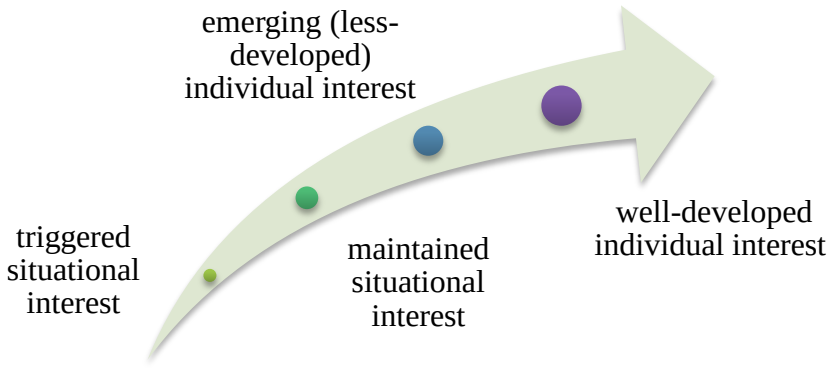
Interest. As pointed out by Dewey (1913), interest is the key component of learning and getting students interested in school subjects is the main goal of schools, a goal at which schools almost always fail. Interest, according to Dewey (1913), "is the sole guarantee of attention; if we can secure interest in a given set of facts or ideas, we may be perfectly sure that the pupil will direct his energies toward mastering them" (p. 1). Therefore, interest is at the heart of the efforts to motivate students to learn (Brophy, 1999).

Interest can be defined as "a psychological state that, in later phases of development, is also a predisposition to reengage content that applies to in-school and out-of-school learning and to young and old alike" (Hidi & Renninger, 2006, p. 111). Researchers have found that interest influences the amount of attention people pay to tasks, people's goals, and levels of learning. Traditionally, interest has been categorized into two types depending on how persistent it is. *Personal interest* refers to a person's relatively enduring "predisposition to reengage particular content over time as well as to the immediate psychological state when this predisposition has been activated" (Hidi & Renninger, 2006, p. 113). This is the type of interest students bring with them to school and is relatively harder to change. Situational interest, on the other hand, is "is triggered in the moment by environmental stimuli, which may or may not last over time" (Hidi & Renninger, 2006, p. 113).

Interest development. According to Hidi and Renninger (2006), interest development follows a four-step path from early (triggered) situational interest to well-developed individual interest (Figure 2). The first stage of interest, *triggered situational interest*, is usually "sparked" by environmental factors, which then turns into *maintained situational interest*, a more focused and persistent interest. At this stage finding meaning in the task help deepen interest, which then becomes *emerging individual interest.* Finally, due more knowledge and value building due to

repeated exposure to the task, a *well-developed individual interest* emerges. It is believed that external stimuli (i.e., instructional conditions and learning environment) can facilitate development of interest. The crucial component in development of interest from early stages of situational interest to later stages of individual interest is finding personal meaning and relevance with a task (Hidi & Renninger, 2006; Hulleman et al., 2008; Hulleman & Harackiewicz, 2009). "Perceiving a topic to be useful and relevant for other activities or life goals (i.e., utility value) predicts both subsequent interest and performance" (Hulleman & Harackiewicz, 2009).

Figure 2. *Interest Development (Based on Hidi and Renninger, 2006)*



Students with a history of low-performance are likely to have low interest and self-efficacy at school subjects. Recent research by Hulleman et al. (Hulleman et al., 2008, 2010; Hulleman & Harackiewicz, 2009) showed that interventions that manipulate students' perceived utility value in school subjects impacts their subsequent interest and performances at school, especially for students with low performance or performance expectations. In a similar vein, Brophy (1999) argued that students need to see the relevance of an activity to their lives and identify themselves with them in order for the learning activities to retain their interest. If students cannot perceive relevance and identify themselves with a task, it is the duty of the

teacher "to mediate the learning in such a way that desired self-relevance perceptions are developed" (Brophy, 1999, p. 79). Self-relevance, therefore, is important in fostering interest and value in learning tasks.

Research on developing interest and utility value. According to researchers (Brophy, 1999; Hulleman et al., 2010), an effective way of ensuring that the students see the relevance of a task for their lives is through giving students opportunities to self-generate relevance and identify themselves with tasks in question. Through generating their own relevance and identification with tasks, it is believed that students "discover the connections between an activity and their lives on their own may be a less threatening way to promote the perception of utility value, and it may therefore be particularly beneficial for students with low performance expectations" (Hulleman et al., 2010, p. 881). Educational research, especially research on transfer, also shows that self-reflection (i.e., reflective teaching), or self-generated connections (i.e., generative methods of teaching problem solving, or meaningful learning) are effective methods of teaching and learning (see "Teaching of problem solving" section for a detailed description "generative methods" and "meaningful learning").

In recent studies, students' utility value perceptions were manipulated to see its effects on their perceived utility value, interest, and performances at school subjects. Hulleman et al. conducted this series of studies (2010; Hulleman & Harackiewicz, 2009). In their first study, Hulleman & Harackiewicz (2009) randomly assigned students taking a high-school science class into one of the two conditions: relevance and no-relevance. In the relevance condition, the students were asked to write an essay on how science could be useful in their lives, whereas the control group wrote about wrote a summary of the material they were learning at the time. At the end of the semester, low-expectancy students in the relevance condition reported more interest in

science and received higher grades at the end of the semester than the low-expectancy control group students. Interest in science at the end of the semester was a significant predictor of future tendencies to select science careers or classes.

In their follow up study (Hulleman et al., 2010), where a similar relevance intervention was used in both a lab setting (study #1) and college classroom (study #2), the authors found similar results. In study #1, students who received relevance intervention on using a simple math technique showed increased interest in the technique than the other students. In addition, participants in the relevance condition were more interested in using the technique in the future than the control condition. In study #2, the authors replicated the study in an undergraduate psychology classroom. The results were very similar to the first study, in that the students in the relevance condition were more interested in the course at the end of the semester and were more inclined to major in psychology. The students with initial low scores or interest in psychology especially benefited from the intervention.

To understand the conditions to nurture interest in educational technology among pre-service teachers, Phillips (2007) conducted a quasi-experimental study where he manipulated meaningfulness and perceived involvement in educational technology lessons. Four classrooms were assigned to one of the conditions: a meaningful lesson (ML-only), involvement (I-only), meaningful lesson and involvement (I+ML) and control. The results of the study showed that the students in ML-only, I-only, or ML+I conditions were found have greater increase on personal interest of technology than the control group, and ML-only groups were better than I-only and control. Similarly, experimental groups showed greater increase in perceived competence of technology than the control group. I+ML group had higher perceived competence than the other experimental groups. The I+ML group also showed greater increases in situational interest in

technology than both I-only, and ML-only. It was seen that meaningfulness of a lesson, and a person's involvement in an activity was a strong predictor of personal and situational interest.

Another study looked at the relationship between relevance and interest was conducted by Shin (2010). Shin conducted a mixed-methods study in order to understand the impact of providing rationales to students, as well as to investigate whether the source of the rationale made a difference on the potential impact. Undergraduate students taking an online introduction to a class on teaching strategies were randomly assigned to one of the 3 conditions: student rationale condition, instructor rationale condition, and a control group who did not receive any rationales. The students in the rationale conditions listened to audio statements about the value of the upcoming lesson at the beginning of each module. Students' self-report perceived value, interests in the course, self-determination, regulation, and their learning of the course (assessed through their final grades in the course) were examined to see the impact of the conditions. The results showed that the students' perceived utility value of the course did not differ for groups. Similar results were found for interest in the course. In terms of final grades, there was a significant difference among the groups, favoring the student rational group over no rationale and then instructor rationale groups.

Summary. Interest is one of the key factors that help predict future reengagement with tasks and performance. Interest, however, has to be developed from early stages of triggered situational interest into later stages of well-developed individual interest. It is also known that utility value (a subcomponent of expectancy-value theory) is a strong predictor of interest development and performance at tasks.

Research has shown that, by manipulating utility value, it is possible to improve value, impact performance, and develop interest. There are not any studies in the domain of game

design that looked at the relationship between theoretical motivational constructs and these tasks. In addition, there are not any studies that investigated the process of interest and value development in a secondary, underlying task (i.e., problem solving and programming), through exposure to a primary task (i.e., game design). Therefore, we do not know if game design tasks are viable platforms to develop interest and value in domains beyond game design itself, namely in programming and problem solving.

Overall Summary

Problem-solving skills are important for young children to develop and be successful in their future careers, and unfortunately, they do not get the necessary opportunities to learn and practice these skills at school settings. Programming, software design tasks, and game design are viable platforms for fostering these thinking skills. Schools often do not provide students with chances to experience hands-on design activities or to solve complex real-life problems. These skills are very important for the future careers of young students, and they need to be provided with chances to practice these skills.

Researchers have identified various methods of teaching problem solving, such as teaching basic skills, teaching by using analogies, generative methods, structure-based methods and teaching metacognitive skills directly. Game design is also a viable and engaging platform to develop problem-solving skills, due to the inherent nature of the game design process. Researchers also argue that game design tasks are promising venues to introduce students to STEM careers and raise interest in STEM fields. Due to a lack of research, however, we do not know if game design tasks are suitable to teach students problem-solving skills.

Students enjoy using computers in school settings, regardless of the context for those tasks, and they especially like game design or programming activities. Although qualitative or

anecdotal reports of student engagement while working on software or game design tasks abound, we do not know if the "motivational" impacts of these tasks actually exist at an empirical level, and if they do, it is unknown in what capacity they work. Considering the main purpose of exposing students to these tasks is to actually "trigger" interest in the underlying thinking skills and programming, it is necessary for research to empirically establish the connection between these tasks and important motivational constructs that predict future choice and performance (i.e., utility value and interest). In this study I aim to provide an empirical test of the cognitive benefits (i.e., problem solving skills) of learning game design, as well as to investigate the extent to which these tasks influence students' interest and utility value in underlying problem solving and programming skills.

CHAPTER 3

Game Design and Learning Intervention

Overview

In this chapter, I will first describe the guiding principles that inspired the structure of the Game Design and Learning (GDL) intervention (i.e., constructionism and creating a "Clubhouse" atmosphere). Then, I will talk about the practicalities of the game design software used at the GDL intervention: Microsoft Kodu. In the remainder of the chapter, I will explain the structure of the GDL curriculum, specific tasks offered during the intervention and rationale behind these tasks in detail.

Guiding Principles of the GDL Intervention

Constructionism. GDL courses, through the process of helping students make games, is a genuine example of Papert's *constructionism*. As a theory, Papert's *constructio**n**ism* is very similar to *constructi**v**ism* in that they both see learning as an active process of building knowledge (Ackermann, 2001; Papert & Harel, 1991; Papert, 1980, 1994; Suomala & Alajaaski, 2002). Maintaining the same basic principles, Papert's constructionism

> adds the idea that this happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity whether it's sand castle on the beach or a theory of the universe. (Papert & Harel, 1991, p. 1)

Constructivism distinguishes between rote and meaningful learning, and it is active construction of learning which leads to meaningful learning. In meaningful learning, acquiring general principles or patterns is the goal (Bruner, 1961; Mayer & Wittrock, 1996; Mayer, 1999), while rote learning is a process of memorization of meaningless knowledge bits. It is argued that

learning is most effective when learners create meaningful and personal artifacts as outcomes of the learning process (Kafai, 1995).

The GDL intervention in this study is an example of a "constructionist" learning environment. During the GDL intervention, students create products (i.e., games) that are personal and meaningful for them. Through this active process of construction, students not only create artifacts that are meaningful for themselves and their peers, but also show persistence in the task due to high levels of interest and personal nature of the creation process.

Guided discovery learning. Research comparing guided versus pure discovery learning methods has indicated that guided discovery learning can be superior to pure discovery learning when teaching thinking skills (Clements, 1999; Kirschner et al., 2006; Mayer, 2004; Palumbo, 1990; Salomon & Perkins, 1987; Woodward, Carnine, & Gersten, 1988). For example, in studies that compared the effectiveness of guided discovery methods to pure discovery methods, the students in the guided discovery conditions performed significantly better on the given tasks in comparison to their peers in pure discovery conditions (Kirschner et al., 2006; Littlefield, Delclos, Bransford, Clayton, & Franks, 1989; Mayer, 2004).

Although in both guided discovery and pure discovery methods learners are expected to construct their own learning, the main difference between the two methods is that in guided discovery learners are guided through this process by getting "hints, direction, coaching, feedback, and/or modeling to keep the student on track" (Mayer, 2004, p. 15). According to Mayer and Wittrock (1996), "[i]n pure discovery, no guidance is provided, whereas in guided discovery, the teacher provides enough guidance to ensure that the learner discovers the rule or principle to be learned" (p. 54). In pure discovery, due to the lack of guidance, children fail to make the underlying connections between concrete tasks (i.e., making games) and abstract

concepts (i.e., problem solving). Early research by Pea and Kurland (1984) also pointed to the need for guidance while teaching students the connection between programming and thinking skills, and pointed to the lack of guidance as one of the reasons why students failed to see the connection between the two.

Overall, a substantial body of literature supports the effectiveness of guided discovery methods over pure discovery methods (Kirschner et al., 2006; Mayer, 2004). Students understand the connections between programming tasks and thinking skills more effectively through guided learning activities are. GDL courses are, therefore, based on these principles of guided discovery learning and daily activities are carefully constructed to scaffold students thinking.

"Clubhouse" atmosphere. Connected tightly with LOGO research, Computer Clubhouses (Kafai et al., 2009; M. Resnick & Rusk, 1996) have introduced underprivileged children to computer technology since 1993. One of the main purposes of the clubhouses has been to encourage "youth creativity, in young people taking on the role of producing rather than merely consuming technology" (Kafai et al., 2009, p. ix). This extra focus on creativity and production through design activities is one of the main elements of these programs that make them engaging for youth and this structure and atmosphere of these after-school programs was a role model for the GDL courses.

Specifically, the GDL courses benefit from four foundational principles of early Computer clubhouses: "a) support learning through design experiences, b) help youth build on their own interests, c) cultivate 'emergent community', and d) create an environment of respect and trust" (M. Resnick & Rusk, 1996, p. 453).

In the GDL, design is at the heart of the curriculum. The main activity of game design gives learners chances to actively engage in problem solving tasks that are meaningful for them.

These design experiences foster students' own game design skills, and they also get chances to practice their programming and problem-solving skills. In addition, students came to GDL with high levels of interest in the course activities because they self-selected to participate. This enables active participation and persistence, even when the tasks are hard. Moreover, learners often collaborate with their peers during the game design process, and this allows for cultivation of emergent communities. During GDL, it has been observed that students often group around game genres and end up collaborating with each other, especially by providing troubleshooting help. Finally, during the GDL courses, the course atmosphere is built around mutual respect and students are given chances to express themselves. Just as it was the case with the clubhouses, at GDL courses the purpose is not "simply dole out praise to improve the "self esteem" of the youth. [We] treat youth more like colleagues, giving them genuine feedback, and pushing them to consider new possibilities" (Resnick & Rusk, 1996, p. 438). The unique structure of the GDL courses gave students enough responsibility for their own actions, while offering support when they needed it.

Built upon these core principles mentioned in this section, GDL courses were designed with the hope to provide students with unique and engaging experiences within a collaborative and respectful community where they can actively construct knowledge, work on personally meaningful tasks, and develop their own thinking skills.

Course Software: Microsoft Kodu

Kodu is a game design software that is specifically designed for children to provide an early entry to computer programming (MacLaurin, 2011). As a game making platform, "Kodu allows users to create their own video games by designing the world, deciding which characters will appear in it, and programming the characters using an easy-to-understand visual

programming language" (Stolee & Fristoe, 2011, p. 99). The software's main purpose is to "help users learn computer science concepts through game creation" (p. 100).

There are numerous reasons why Kodu was selected over other game design or animation software. First, as opposed to other software, Kodu provides a three-dimensional environment (3D), and it is visually more appealing for students (Figure 3). In addition, the created game environments can be expanded to replicate big, endless worlds that can be found in most commercial games, as opposed to two-dimensional and limited worlds provided by alternative game design software.

Figure 3.*Screenshots from Microsoft Kodu: Visual Programming - The text within the visual is not meant to be readable, and is meant for reference only.*

Secondly, Kodu uses a programming language structure that is core to most common programming languages (Stolee & Fristoe, 2011). Due to this structure, it lets users to explore some important basics of computer programming, such as variables, Boolean logic, objects, and control flow. In Kodu,

> [u]sers can program each character (e.g., a fish, cycle, apple, tree) individually, and the programming defines how it interacts with the world, much like an intelligent agent. The programming takes place on pages, which define different states for the character. A character may contain up to 12 pages of programming, and can maintain its state and

> control flow by switching between pages. Each page contains a set of rules, where a rule is analogous to a statement in typical programming languages. Each rule is in the form of a condition and an action, which form a WHEN + DO clause, that is, when condition, do action. If the condition is satisfied, then the action is performed….There are over 500 tiles that can be used to compose rules in the Kodu Language. (Stolee & Fristoe, 2011, p. 100)

Third, Kodu's flexibility in its language is better for creating simulations, as opposed to games, where the simulation is allowed to run by itself and count user-defined events. For example, the students created a predator-prey relationship simulation and programmed Kodu to count the populations of every type of character in the world. This adds a great amount of power to Kodu's educational potential, especially considering the main purpose of teaching problem solving skills and the analysis of systems and structures.

GDL Curriculum Structure

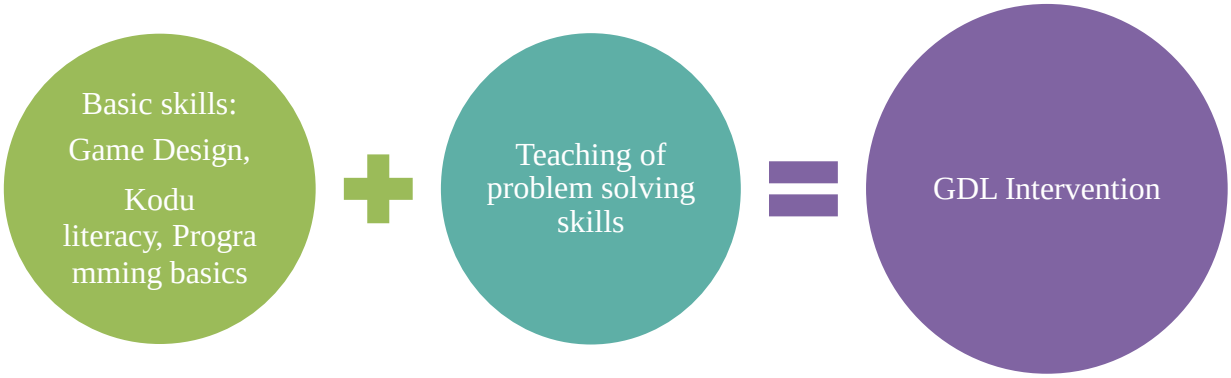
The GDL curriculum is engineered to teach students how to design games using commercially free software, Microsoft Kodu. The courses have two overarching goals: (a) teaching students how to use the software, and (b) showing students the connections between game design and problem solving (Figure 4).

The first goal was achieved through game design activities based on fostering students' Kodu knowledge. These teaching activities were followed by hands-on sessions in which students learned to create familiar classic games, such as Pac Man, Super Mario Brothers, or generic tower defense games.

The second goal was achieved by providing students with hands-on opportunities to solve problem within assigned scenarios, and reverse engineering these scenarios to create games in

Kodu. During this process the instructors were available to answer questions from the students about, for example, how to troubleshoot a certain line of code to fix their games or getting help on aesthetics.

Figure 4. *Two-step Structure of GDL Curriculum*



This two-step approach to configuring GDL activities, namely offering problem solving tasks only after students mastered game design and programming, is based on two theories: teaching basic skills (i.e., automaticity and constraint-removal) (Mayer & Wittrock, 1996) and chain of cognitive changes (Dalbey & Linn, 1985; Linn, 1985; Mayer & Fay, 1987).

According to theories of automaticity and constraint-removal, mastery of low-level cognitive skills makes it easier for people to allocate more time and energy in solving problems requiring high-level cognitive skills. The GDL curriculum focuses the early activities on teaching students low-level skills necessary to master the game design software, the game design process, and programming basics required to design games in Kodu.

Chain of cognitive changes model (Mayer & Fay, 1987) proposes that while learning a programming language learners go through steps starting from low-level, basic components and moving to higher-level, advanced components. For example, while learning LOGO, the model predicts that learning of basic programming syntax will happen first, and it will be followed by

semantics (i.e., learning to think within the domain of programming), and finally, transfer (i.e., learning to think outside programming). More importantly, students' success in later steps is contingent upon their mastery of the prior steps. Similarly, in the GDL curriculum, the mastery of the basics of the game design software and programming basics are given priority, in order to make it possible for students to move to more advanced levels. In a similar vein, the activities requiring advanced problem solving skills are introduced only after the students master basic requirements so that they can think of outside the game design context and develop more generalized skills.

GDL Activity Types

As mentioned in the previous sections, the activities used during GDL intervention are created based on multiple different theories (i.e., theories of teaching problem solving), and also are designed to teach students game design, programming basics, as well as to provide practice in specific problem skills (Table 2). These activities usually take 3-4 hours to complete. Depending on the time available, these activities take up the whole day, or if there is more time available the students are given time to creating more complex designs, or work on their own games.

Table 2. *Progression of Activities and Specific Activity Types Offered at the GDL*

Progression	Activity name	Type	Purpose	Problem solving involved	Duration
Activity #1	Apple hunter	Game design	Teach basics of Kodu + Programming	SYS, DM, TS	3-4 hours
Activity #2	PacKodu	Game design	Teach basics of Kodu + Programming	SYS, DM, TS	3-4 hours

Table 2. (Cont'd)

Activity #3	Kodu Adventure	Game design	Teach basics of Kodu + Programming	SYS, DM, TS	3-4 hours
Activity #4	SimSchool	Problem solving (simulation)	Teaching problem solving	SYS, DM, TS	3-4 hours
Activity #5	Predator-Prey	Problem solving (simulation)	Teaching problem solving	SYS, DM, TS	3-4 hours
Activity #6	Fix a broken game	Troubleshooting	Teaching problem solving	SYS, DM, TS	3-4 hours
Activity #7 (optional)	Create your own game	Free design	Teaching problem solving	SYS, DM, TS	3-4 hours

**SYS = System analysis and design, DM = Decision making, TS = Troubleshooting*

In the remainder of this section, I will describe one example activity from each category and the specific theories they are based on, and in which problem types they provided practice (See Table 3). The specific activities that I will describe are (a) *game design,* (b) *problem solving,* (b) *troubleshooting,* and (d) *free design.*

Table 3. *GDL Activity Types and Their Alignment with Course Goals and Problem Solving*

Category	Purpose	Subtasks	Problem solving
Game Design	• Game design • Programming • Microsoft Kodu	• Reverse-engineer game from given GRASPS • Create flowchart	• System analysis and design • Troubleshooting • Decision making

Table 3. (Cont'd)

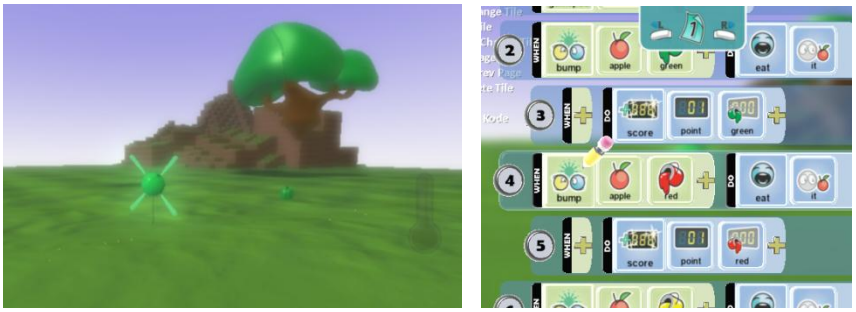
Problem solving	• Problem solving skills • Familiarize with complex problems • Programming • Microsoft Kodu	• Analyze a given problem scenario • Find solution to the given problem • Understand problem patterns from data and graphs • Create simulation of the problem scenario • Create flowchart of the scenario	• Metaskills (plan, devise solution, execute, evaluate) • Recognize patterns • System analysis and design • Troubleshooting • Decision making
Troubleshooting	• Troubleshooting • Familiarize with complex problems • Programming • Microsoft Kodu	• Find inoperable parts of a game and fix them • Create a flowchart of the game • Create GRASPS of the game	• Troubleshooting • System analysis and design • Recognize patterns • Decision making
Free design	• Game design • Programming • Microsoft Kodu	• Create flowchart of the game • Create GRASPS of the game • Troubleshoot when necessary • Make decisions based on software limitations	• System analysis and design • Troubleshooting • Decision making

Game design activities. The purpose of the *game design activities* is to help students learn the basics of game design, programming, as well as basics of Microsoft Kodu. Therefore, students are provided with many opportunities to create popular games using Kodu, while learning about the intricacies of the game design and programming, and also master Kodu.

During game design activities students' knowledge of game design and programming is scaffolded through three specific tasks: creation of games (from simple to more complex), working on identifying elements of games, and creating flowcharts of games.

An example game design activity that the students do on the first GDL session is a game called, "Apple Hunter." This activity is designed to teach students both the basics of Microsoft Kodu, as well as the basics of game design (See Table 3). Apple Hunter is a very simple game where the goal is to create a playable character, and the player wins the game when the character eats five green apples and reaches five points (Figure 5).

Figure 5. *Screenshots from Apple Hunter - The text within the visual is not meant to be readable, and is meant for reference only.*



The instructional sequence for a sample game design activity includes instructor-led sections, as well as sections where students work individually (Table 4).

Table 4. *Instructional Sequence for a Sample Game Design Activity (Apple Hunter)*

Subtask	Instruction	Length	Ideal Location
Introduce GRASPS	Led by teacher, solicit students response for popular games' GRASPS	30 minutes	Class with projector
Introduce Apple Hunter	Demo game creation by the instructor – Step by Step: Instructor creates, students create each step	1.5 hours	Class with projector + Computer Lab
Apple Hunter Flowchart	Instructor-led session on: Analyzing flowcharts, Analyze Apple Hunter flowchart Student Activity: Improve AH Flowchart	1 hour	Class with projector

The session begins with a focus on the basics elements common to most games, namely **G**oals, **R**ules, **A**ssests, **S**paces, **P**lay mechanics, **S**coring (GRASPS) (Table 5). By generating the GRASPS of popular games with the whole class's participation students get a chance to look from a game designer's perspective to the games that they play for the first time. At this stage, the instructor solicits answers from the students in order to raise their awareness as to what are the necessary elements of games.

Table 5. *GRASPS for Apple Hunter Game*

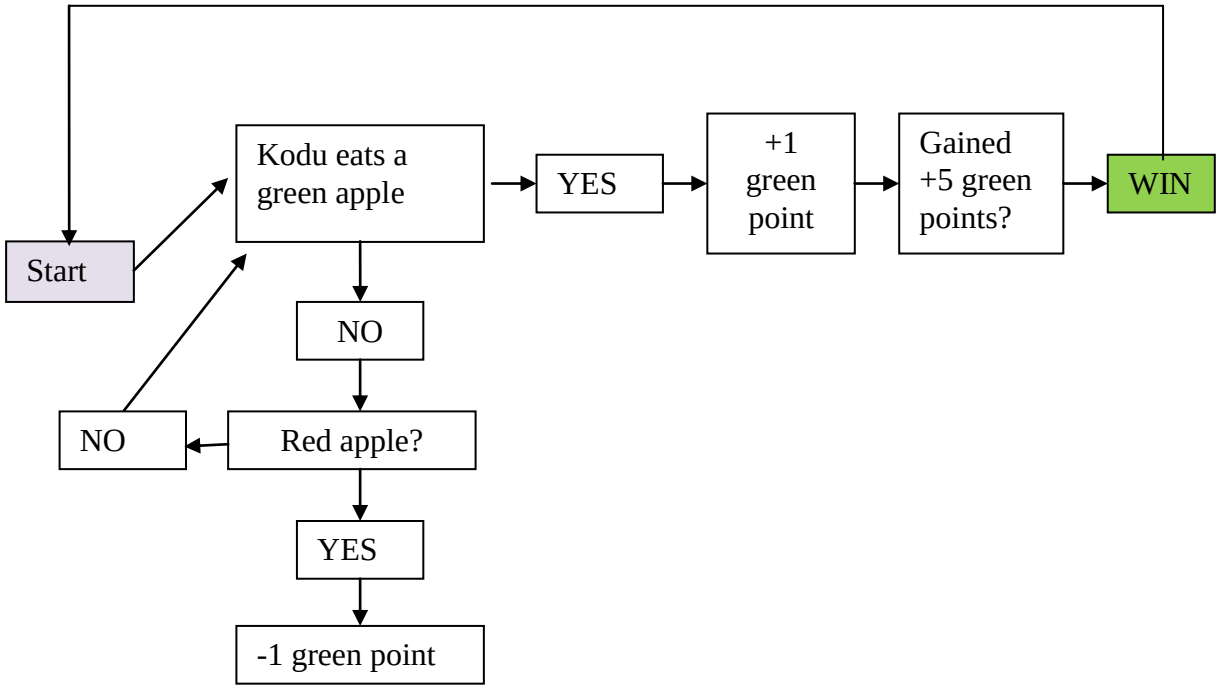
Apple Hunter	
Goals	Eat 5 green apples
Rules	Do not eat red apples *(optional: character slows down after eating red apples)* *(optional: in 20 seconds)*
Assets	Kodu Apples Tree
Scoring	Green apple = +1 green point Red apple= -1 green point
Play mechanics	Wander around, look for/avoid apples, bump to eat
Spaces	Walled open world

Next, students are provided with GRASPS of Apple Hunter and are asked to reverse engineer to create this game. At this stage, the instructor gives out the worksheet, and quickly goes over the GRASPS of the game with students to check for understanding. After this, since this is the first class and students do not have prior experience with Kodu, the creation of Apple Hunter is done as a whole class. The purpose here is to teach students the basics of Kodu, and also the basics of programming and game design.

To create Apple Hunter, first a playable character is created and programmed to move with the help of the keyboard, and then a tree that generates apples at random is created, finally the character is programmed to eat green apples and score +1 point. This process teaches the students the basics of the process. During the rest of the class, the students work on their own to improve the game, or recreate it from scratch on their own. Instructors provide support (i.e. programming, troubleshooting) if needed.

Activities in the *game design* category continue with creating flowcharts of games the students will create. Flowcharts are representations of games depicted as systems. First the students are given the flowchart of the game, Apple Hunter in this case, (Figure 6). After giving students a couple of minutes to analyze the flowchart, the instructor asks the students questions such as "how many points do we need to win the game?" to check for students' understanding. Finally, in order to give students chances to practice how flowcharts work, the instructor asks them how they would change the flowchart if, for example, the red apples ended the game immediately. This way they get a chance to practice reading flowcharts. In following lessons, following a similar instructional sequence, students are scaffolded toward creating their own flowcharts, by gradually giving them less complete charts, and finally giving them blank sheets and asking them to create from scratch.

Figure 6. *Example Flowchart Provided by the Instructors for Apple Hunter Game*



During game design activities students the practice specific problem-solving skills that are investigated in this study, as well: system analysis and design; decision-making skills; and troubleshooting. For example, through creating or analyzing flowcharts of games or while reverse-engineering a game from its given components, students practice system analysis and design skills.

The game design process is also a natural context for student to practice troubleshooting skills. For example, if a character won't work or a tree won't produce apples, the students will need to carefully look at the codes of their games in order to find a line of code that requires fixing in order to produce the desired outcome. Additional opportunities arise in the game design tasks for student to practice their decision-making skills. For example, students need to decide the specific elements to add to their games within the constraints of GRASPS (Table 5). A student might have to choose among deciding to add red apples to the game, or to deduct points

when his/her character eats them (as opposed to the green apples which are associated with gaining points), or to create a countdown timer and turn the game into a time-based race, or even to add another character into the game to compete against. All of these decisions significantly change their games, and each choice presents the opportunity to practice decision-making.

Problem-solving activities. In the second half of the GDL intervention the focus is on instructional activities centered on helping students to see the connection between the game design and problem solving tasks. As stated earlier, this phase occurs after students have mastered the game design software, the basics of game design, and programming. During this second phase students not only get chances to practice their general and specific problem solving skills, but also get acquainted with important metacognitive and basic skills underlying the problem solving process (Table 6). More importantly, during this phase, the instructional activities are based on theories of teaching problem-solving described in Chapter 2.

Table 6. *Instructional Sequence for a Sample Problem Solving Activity*

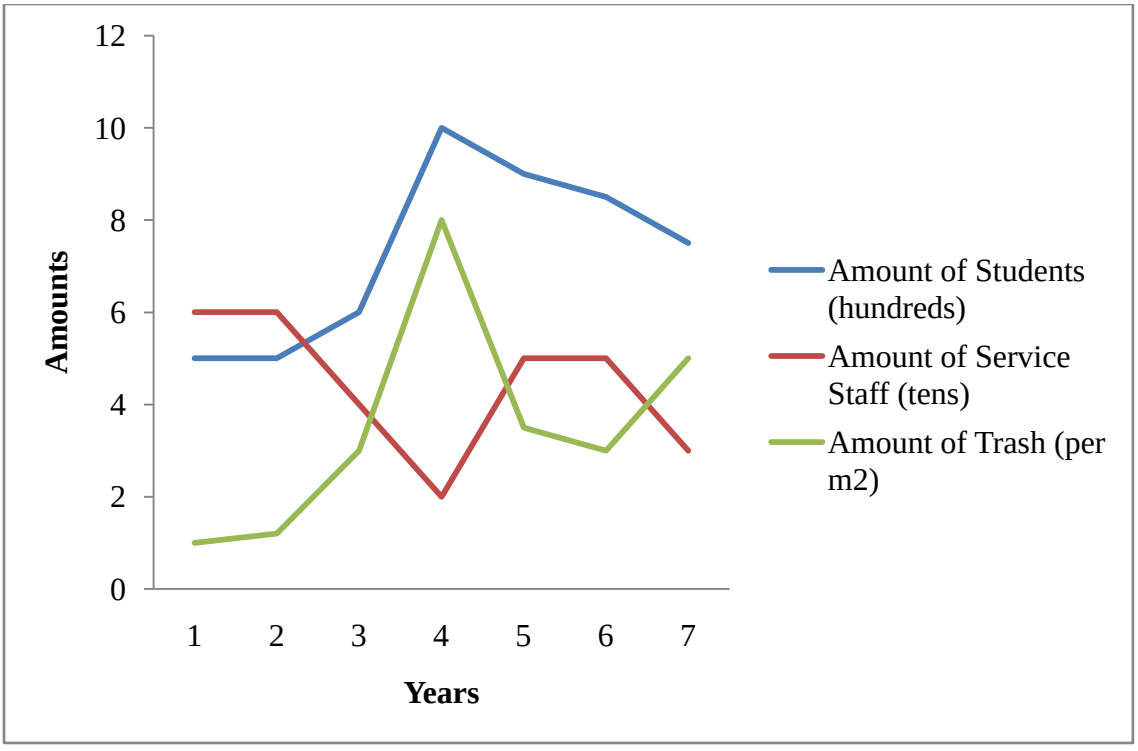
Subtask	Instruction	Length	Ideal Location
Introduce Problem Scenario	Led by teacher Present problem scenario Solve with the class	45minutes	Class with projector
Recreate Scenario in Kodu	Demo game creation by the instructor – Step by Step: Instructor creates, students create each step (*first activity only*) Individual student work to complete the scenario in Kodu, improve	1.5 hours	Class with projector + Computer Lab
Create Flowchart	Students work on creating the flowchart for the problem scenario based on their Kodu simulations Instructors help as needed	1 hour	Class with projector

An example activity that can explain general process of a problem solving activity is called “SimSchool.” The premise of this activity is to introduce students to the connection

between game design and problem solving. To do this, students are taken through a multistep process by the instructor: (a) the students are first introduced to a complex problem, (b) are guided in ways to solve this problem (using methods of teaching problem solving), and, finally, (c) are asked to recreate the simulation in Kodu.

In SimSchool, students first receive the background of the problem-solving scenario: "There is a trash problem at a school, and you need to understand the source of this problem and devise a solution, and recreate your solution in Kodu." During the first step, the students receive data and graphs regarding the issue (Figure 7). It is at this first step, through instructor guidance, that the students are introduced to basics skills underlying problem solving.

Figure 7. *SimSchool Problem Scenario – Data Sources*



During this activity specific methods of teaching problem solving got used: *teaching basic skills* and *teaching metacognitive skills directly* (Mayer & Wittrock, 1996). Teaching of

basic (i.e., low-level) skills involve teaching low-level cognitive skills to a degree where they are automated so that while solving complex problem students' cognition gets less taxed. Learning game design, programming and mastering Kodu during the first phase of the GDL can be an example of this.

As for teaching of metacognitive skills during the first step of SimSchool, students learn how to look at the data and the graph to understand the source of the problem. During this step instructor provides guidance (i.e., guided discovery learning) without instructing students on the right answer. During this step, also, the students receive instruction on some basic metacognitive skills required to solve any complex problem. Specifically, the direct instruction focused on four steps of solving problems identified by Polya (1957): (a) understand the problem, (b) devise a plan, (c) carry out the plan, and (d) look back. Students are especially encouraged to take their time, look at the patterns in the data to understand the source of the problem.

In the next step, they are asked to plan (i.e., create flowchart) their simulation in Kodu (based on the problem scenario). Finally, they are asked to create the game and run and see if the simulation replicates the original scenario. This SimSchool (Figure 8) process maps perfectly onto both Polya's four steps, and the method of teaching metacognitive skills directly. The SimSchool activity allows for student practice employing the metacognitive skills underlying complex problems.

Figure 8. *A Screenshot of SimSchool Simulation - The text within the visual is not meant to be readable, and is meant for reference only.*

It is also during the second phase of the GDL intervention that the *use of analogies* is implemented, another effective method of teaching problem-solving (Gick & Holyoak, 1980; Mayer & Wittrock, 2006). The purpose of using analogies is to help students understand to look for patterns when they face with problems, and to recognize when they can use strategies they already know to solve the novel problems.

The GDL used analogies by providing students with scenarios that are different on the surface, but are essentially built on same underlying principles. Once students had solved the SimSchool scenario, students were given another scenario called Kodu EcoSystem. In the Ecosystem scenario, the relationship between objects in the system is similar to the relationship in the SimSchool. In SimSchool, service staff clears out trash, and student number and service staff number has an inverse relationship with the amount of trash (Figure 8). Similarly, in Kodu EcoSystem, the three living organisms have a relationship that impacts the population numbers of each other.

In the ecosystem scenario, just as with the SimSchool scenario, the students are guided by the instructor to make sense of this problem, devise a solution, and replicate it in Kodu. By

working on analogous problems, the Ecosystem scenario activities provided students with opportunities to work with analogies and to see patterns in seemingly complex problems. During this process of solving the Ecosystem scenario, students master the game design software, while also working on reverse engineering a very complex problem. Finally, the scenario provided a context for hands-on practice with a fun activity to solve a complex problem.

Within the context of the SimSchool and Ecosystem scenarios, another very effective method of solving problems, *structure-based* methods is also used. In structure-based methods students work with concrete objects and hands-on tasks, which are considered to be effective ways of teaching for understanding (Mayer & Wittrock, 1996; Moreno & Mayer, 1999). In structure-based methods, teachers can, for example, use beads and sticks in mathematics (concrete objects) to teach simple computation problems (abstract rules). In using Kodu to recreate a problem scenario, the GDL intervention exemplified structure-based methods of teaching problem solving. Designing games is already a hands-on activity, and can be considered as a more contemporary example of structure-based methods.

Finally, during the process of recreating problem scenarios in Kodu, *generative methods* of teaching problem solving was also implemented. Generative methods (Mayer & Wittrock, 1996) are designed to push students to generate relationships between their own experiences and the target information during learning activities. The problem tasks used in the GDL intervention gave students the opportunity to relate the problem scenarios to their own experiences in two ways. First, the scenarios are selected from contexts that are similar to students' daily life experiences (e.g., SimSchool). Second, during the recreation process, students design worlds that they imagine. For example, students sometimes changed the student-staff problem into an

employee-boss problem. It is this process of personalization and generation that results from the use generative methods of teaching problem solving.

Troubleshooting activities. Troubleshooting tasks are included placed in the GDL intervention both to give students recurring chances to practice this vital skill, as well as to help students understand the importance of troubleshooting during the game design process. The troubleshooting tasks are offered in the second phase of the GDL intervention, after students master the game design software, basic game design, and programming skills.

An example of a troubleshooting task is to provide students with a broken game, and then ask them to analyze the game to find the wrong or missing codes (Table 7). In the next step, they are asked to fix these problems in order to make it a working game again.

Table 7. *Instructional Sequence for a Sample Troubleshooting Activity*

Subtask	Instruction	Length	Ideal Location
Introduce the broken game	Teacher shows the broken game Solicit responses from class to troubleshoot the game	30 minutes	Class with projector
Fix the game	Students work individually to work on fixing the game Improve game after fixing (make it your own)	1.5 hours	Computer Lab
Flowchart	Create the flowchart of the game	1 hour	Class with projector

Finally, in the last step, the students are asked to create a flowchart and GRASPS of the game, so that they also get a chance to practice system analysis and design skills. Troubleshooting processes also inherently require students to make decisions: what to fix and which problem has the priority. Due to the immediate feedback (i.e., the effect of the changes made to the code can be seen by running the game instantly), students get to practice their

troubleshooting skills in a fast manner. Also, since the game design environment provides the students with a safe sandbox-like environment, the students develop in their confidence in troubleshooting.

Free design activities. During the final sections of the GDL intervention, students were given chances to work on creating a game of their own. Having mastered the game design software, game design in general, programming and problem-solving skills within the earlier phases of the intervention, students got a final chance to create a personally and socially meaningful artifact in the spirit of constructionism.

Free game design activities also provide an important way to check student understanding, because the students get a chance both test the boundaries of their skills and the game design software. The GDL intervention culminates with student presentations of the games they have developed.

Summary

The GDL intervention was created and offered for the purpose of teaching young children game design, programming and problem-solving. Having multiple goals, inevitably, required engineering of a multi-faceted intervention, incorporating multiple instructional techniques, and activities with different objectives.

The intervention was structured in two phases. During phase one, the goal was to teach students how to Microsoft Kodu and how to design games. This was achieved through guided sessions where students worked on analyzing, planning (i.e., flowcharts) and designing popular games. During this phase, students also got chances to practice their system analysis and design, troubleshooting and decision-making skills, because these skills were embedded in the game design process and naturally emerged.

After teaching the students basics of the game design process, in the second phase, the goal shifted more toward showing students the connection between game design and problem solving. For this purpose, the students received instruction on metacognitive skills underlying problem solving, worked on analogies, solved problems and recreated problem scenarios in Microsoft Kodu. The final step where they recreated the simulations in Kodu especially gave students the hands-on experience in solving complex problems. Within the scope of the GDL intervention, through game design tasks students found many hands-on opportunities to practice their system analysis and design, decision making, and troubleshooting skills.

CHAPTER 4

Methods

Research Questions and Research Design

Research question 1. *Do students attending the GDL courses show significantly higher gains in general and specific problem solving skills (i.e., decision-making, system analysis and design and troubleshooting) compared to the control group?*

In order to understand whether learning game design at the GDL intervention affect students' problem solving skills, interest, and utility value perceptions, I ran a quasi-experimental research (Table 8) where the problem-solving gains of students who attended the GDL intervention were compared to a control group who did not participate in the intervention. The inclusion of a control group was especially important to control for the effects of testing. *Testing* refers to the probability that students who take a test twice would score higher in the post-test by a simple function of familiarity with the test (Campbell & Stanley, 1963). Also, *reactivity,* "process of measuring may change that which is being measured" (p. 9), is a possible cause of change in students' problem solving skills: the mere fact that students had received the problem-solving test might benefit them differently. Therefore, I gave the test of problem solving to a control group to account for these threats to internal validity. The study's design in assessing problem solving skills was nonequivalent control group design (Campbell & Stanley, 1963).

Table 8. *Research Design for RQ 1 (based on Campbell & Stanley, 1963, p. 40)*

Selection	Condition	Pretest	Intervention	Posttest
Self-selected	Experimental	Yes	Yes	Yes
Self-selected	Control	Yes	-	Yes

Research question 2. *Are there differences among the different experimental sites in terms of the gains students show in general and specific problem solving skills?*

Because GDL courses were offered at different sites, and students self-selected to participate in the courses, I also looked at the differences between the experimental sites. The reason for this research question is to understand if the intervention led to significant changes regardless of difference between sites. The study design for this question was pretest-posttest design (See Table 9).

Table 9. *Research Design for RQ 2-6 (based on Campbell & Stanley, 1963, p. 8)*

Selection	Condition	Pretest	Intervention	Posttest
Self-selected	Site 1	Yes	Yes	Yes
Self-selected	Site 2	Yes	Yes	Yes
Self-selected	Site 3	Yes	Yes	Yes
Self-selected	Site 4	Yes	Yes	Yes

Research question 3-6. Similarly, because only the experimental sites received the motivation surveys, I used a similar design to RQ2 for research questions three, four, five, and six.

RQ 3a. Are there changes in students' perceived utility value in game design, programming and problem solving activities due to participating in GDL courses?

RQ3b. Are the changes different for different sites?

RQ 4a. Are there changes in students' situational interest in game design, programming and problem solving activities due to participating in GDL courses?

RQ4b. Are the changes different for different sites?

RQ 5. What is the relationship between students' perceived utility value of game design, programming and problem solving and their emerging individual interests in these domains, controlling for their perceived utility values at the pretest?

RQ 6. What is the relationship between students' perceptions in motivation constructs (i.e., situational interest, utility value, and emerging individual interest in game design, programming and problem solving) and problem solving skills at the posttest, controlling for their performances and perceptions at the pretest?

Research question 7. *Does the relevance manipulation given in the middle of the GDL courses have an impact on students' problem solving skills, situational interest, perceived utility value, and emerging individual interest in game design, programming, and problem solving?*

For research question seven, I used a true experimental design. In true experimental designs, the students are randomly placed into experimental and control conditions (Table 10). The students who attended the GDL courses were randomly split into two groups to understand the impacts of a simple writing task (relevance intervention) on their motivational outcomes (more detail can be found in procedures).

Table 10. *Research Design for RQ 7 (based on Campbell & Stanley, 1963, p. 40)*

Selection	Condition	Pretest	Intervention	Posttest
Random	Experimental	Yes	Yes	Yes
Random	Control	Yes	-	Yes

Participants

Data for this study come from four different groups of middle school students (n = 73) who attended the GDL courses that were offered during summers of 2011, 2012, and fall of 2012 at various sites in Lansing, Michigan and Istanbul, Turkey (See Table 11). Although within each location the groups were homogenous in terms of student characteristics (i.e., SES, and age), due to the self-selected nature of participation, potential differences between the groups are expected.

Table 11. *Summary of GDL Implementation Sites and Sample Sizes*

GDL Site	*n*	Age
Işık Summer 2011	18	12.5
Işık Summer 2012	11	12.3
ITEC LCC GATE	7	11.6
Doğa Schools, Istanbul	13	12
Control Site, Istanbul	24	13.4

Participant Flow

For research questions one and two (i.e., the changes in problem solving skills), the analyses were conducted using data from 73 students (out of a total of 76 students) who attended the GDL courses.

For research questions three through seven (i.e., involving motivational constructs), data was collected at only two (Doga, LCC) of the four experimental sites, yielding a possible sample of 20 possible participants. Out of the 20 students, two participants were removed from further analyses because both students selected the highest possible ratings for all pre- and post-survey items. In addition, one student did not complete the post-survey, and removed from overall analyses. One case was identified as an outlier through calculating the *z* scores for all variables, and identifying the cases that had a *z* score greater than three, resulting in a final sample size for these questions of n = 16.

Finally, for research question 7, data from four students could not be used because some students were not present during the day of relevance intervention (missing at random). This resulted in a final sample size for this analysis of n = 12 (Table 12).

Table 12. *Participant Flow by Research Questions*

Research Question	*n*
1 & 2	73
3-6	16
7	12

Sampling Procedures

The students who participated in the GDL courses were self-selected. The students learned about the courses through announcements made at their schools or at institutions where they were attending other after-school courses. Interested students enrolled in the GDL course, including the control group, which was composed of the students who wanted to participate in the course at the future implementation. Among the enrolled students, data was not collected from the students who did not want to participate in the research (i.e., take the problem solving test and the motivation surveys). The students received small gifts (e.g., Michigan State University bracelets, pens, etc.) as a compensation of their time for taking part in the research.

Research Sites

The research sites were selected because of the personal connections the lead researcher had with the administrators of the sites, who also indicated interest in providing their students with such an alternative extracurricular activity.

Ayazağa Işık Private School, Istanbul. The first implementation of GDL courses took place in Istanbul, Turkey during the summer of 2011. During the first implementation, 19 middle school students (6 female, 13 male, age average = 12.5) joined the course for nine days between June 27 and July 8. Data from one student was removed from the data analysis after it was identified as an outlier as a result of a *z*-score analysis.

The second iteration of GDL courses took place at the same school with a similar group of students during the summer of 2012. In this iteration, 12 new students (11 male, 1 female, age average = 12.3) attended GDL courses. One student from this cohort was removed from further data analyses after it was identified as an outlier as a result of a *z*-score analysis.

All students at the Isik site came from middle or upper-middle class families living in Istanbul, Turkey. The course at these sites lasted for approximately 40 hours (5 hours each day, 8 full days) in each iteration.

ITEC LCC Gifted and Talented Education (GATE) Saturday School, Lansing, MI. Between October 20 and November 17 a GDL course offered through GATE Saturday School of Lansing Community College. Although a total of 15 students joined in the GDL courses, only 7 students (1 female, 6 male, age average = 11.6) agreed to participate in the research.

The course lasted for 5 sessions (3 hours each) for a total of 15 hours. The students were coming from upper middle class families living in Lansing, MI. Since the course length was shorter than the Isik school courses, curriculum was adapted to keep key activities on problem solving.

Doğa Private School, Istanbul. Between November 24 and December 15, 14 students attended the GLD course offered. There were 11 male and 3 female students, and the course lasted for 4 sessions (4 hours each) for a total of 16 hours. Since the course length was shorter than the Isik school courses, curriculum was adapted to keep key activities on problem solving. The students' age average was 12, and they were coming from upper middle class families living in Istanbul, Turkey. One student's data at this site was identified as an outlier (*z* score analysis) and removed from further analyses.

Control group. From the Doğa Private School a group of students who signed up for a summer GDL course (n = 24) were recruited to be the control group, and were given the problem-solving test twice, fifteen days apart. The students' age average was 13.4, and they were coming from upper middle class families living in Istanbul, Turkey. There were 12 females and 12 males at this site, and the age average was 13.4.

Procedures

As indicated earlier in the chapter, in order to answer the research questions three different designs were used: 1) nonequivalent control group design (RQ1), 2) pretest-posttest design (RQ2-6), and true experimental design (RQ7).

One of the main purposes of this study was to measure the students' initial and final problem-solving skill levels, as well as the overall progress they showed as indicated by the difference between the two. For this reason, the students who participated in the study completed the PISA problem-solving assessment at the beginning and end of the GDL courses.

In addition to the problem-solving assessment, the students who participated in the study at the LCC and Doga sites also filled in the motivation survey (i.e., situational interest and perceived utility value of game design, programming and problem solving) both at the beginning and end of the course. These students also filled out an emerging individual interest survey for game design, programming, and problem-solving tasks at the end of the last day of the course.

In order to test the effectiveness of a utility value manipulation (i.e., relevance intervention), in the middle of the GDL courses (middle point varied depending on the overall course length at each site) students were randomly assigned one of the two conditions: relevance (relevance) and no-relevance (control) condition for a brief period of time. The summary of the procedures can be seen in Table 13.

Table 13. *Summary of the Procedures Used*

Time	Test	Sites
First Day	PISA Problem solving test	All
	Situational interest of game design, programming, problem solving	Doga and LCC
	Utility value of game design, programming, problem solving	Doga and LCC
Mid-course	Relevance intervention	Doga and LCC
Final Day	PISA Problem solving test	All
	Situational interest of game design, programming, problem solving	Doga and LCC
	Utility value of game design, programming, problem solving	Doga and LCC
	Emerging individual interest of game design, programming, problem solving	Doga and LCC sites

Independent Variables

GDL intervention. The main intervention of interest in this study was the GDL courses. GDL courses, as described in Chapter 3, were designed to teach students game design and also show the students the connection between game design and problem-solving tasks. As it can be seen in Table 14, GDL was offered at different lengths at each site.

Table 14. *Length of GDL Intervention by Site*

Site	Sessions (times)	Session Length (hours)	Course Length (hours)
Işık Summer 2011	8	5	40
Işık Summer 2012	8	5	40
LCC GATE	5	3	15
Doğa Schools, Istanbul	4	4	16
Control Site, Istanbul	0	0	N/A

Although offered at different lengths, the content of instruction at each GDL site was the kept constant by only offering more *free design* activities at the sites where the course length was longer. The main activities that were offered at each site were *game design, problem solving, troubleshooting* and *free design.* The nature and purpose of these activities were discussed in Chapter 3. A breakdown of tasks by each site can be seen in Table 15.

Table 15. *Activity Type Length by Site*

Site	Game Design (hours)	Problem Solving (hours)	Troubleshooting (hours)	Free Design (hours)
Işık Summer 2011	10	10	5	15
Işık Summer 2012	10	10	5	15
LCC GATE	6	6	0	3
Doğa Schools, Istanbul	6	6	0	4
Control Site, Istanbul	0	0	0	0

As is shown in Table 15, although the LCC and Doga sites did not receive specific tasks for troubleshooting due to time constraints, the regular task of game design incorporates a significant amount of troubleshooting and did not need further time devoted to them.

Instruction was given at the beginning of each task for a minimal amount of time, and most of the class-time was devoted to students working on game design or problem-solving activities. The researcher gave all instruction in order to keep instructor effect constant. After the students were introduced to their tasks, the rest of the class time the students worked on creating

their games (according to the given tasks), and both the lead instructor and two other instructors provided individual support when the students asked for help.

GDL classes were offered in computer labs at the research sites (Figure 10). The computers were all Windows PCs with Microsoft Kodu installed. Since XBOX controllers can be used on PCs to design Kodu games, students were given the option to bring their controllers if they had one. The experience of game design, however, was not impacted by whether the students designed with controllers or a standard keyboard-mouse set. During the intervention, each student received one PC and worked individually, although students were not discouraged from helping each other. All instructions were given in English at all sites.

Figure 9. *An Example Classroom Setting Used During GDL Intervention*

Relevance intervention. Similar to previous research by Hulleman et al. (2010; Hulleman & Harackiewicz, 2009), in order to test whether manipulating perceived utility value had an impact on students' performance and interest in problem solving, in this study students were assigned into a utility value manipulation (i.e., relevance intervention) condition.

Specifically, at Doga and LCC sites, at the midpoint of the course the students were randomly divided into two groups. The students were assigned to either the relevance (experimental) or no-relevance (control) conditions. All students followed the same curriculum

and attend the same activities throughout the camp, apart from the relevance intervention on this day.

The students in both conditions received a paper with instructions about the writing assignment adapted from Hulleman et al. (2010), and they were provided with a blank space to write their paragraphs at the bottom of the paper. For this intervention, the students were asked to turn off their monitors, put away their keyboards, and use the desks in the computer lab. The students in the relevance condition received instructions on writing a short essay relating the problem solving skills they were learning at the GDL courses to their future school and real lives:

> …Think about all of the problem solving activities we have been doing in this camp. To convince one of your friends to join the camp next year, write a short essay (1–3 paragraphs in length) briefly describing how the problem solving skills you are learning here are useful to your own school and future work life. You can especially think about how the *system analysis and design*, *troubleshooting* and *decision-making* skills that you are learning at this camp through using Kodu and designing games will be helpful in school, and maybe in your job in the future. Feel free to give examples.

During the same time period, participants in the control condition received instructions on writing a paragraph describing their favorite digital game. The students were asked to keep their work secret in order to prevent them from seeing the other writing task (i.e., relevance task). The instructions for essay were:

> …Think about your favorite game. Write a short essay (1–3 paragraphs in length) describing it. You can especially think of what you like the best about this game (any specific parts of the game).

After students finished writing their essays, the researchers collected the essays, and the students in both groups continued following the same GDL curriculum until the end of the camp without any other interventions or group-specific changes.

Sites. Another independent variable was the site effects. As mentioned previously in this chapter, since the students self-selected to be part of this research, site was used as a between-subjects independent variable in analyzing the research questions.

Dependent Variables

General and specific problem-solving skills. In this study, problem-solving skills were measured by 2003 PISA (Programme for International Student Assessment) Problem Solving Assessment. The test was originally developed collaboratively by OECD (Organisation for Economic Co-operation and Development) member countries to assess and compare 15-year-old students across the globe for their readiness for the challenges of the tomorrow's world (OECD, 2005a). The test-retests reliability of the test was acceptable (α = .645).

The student answers were graded as 0 (incorrect), 1 (partially correct) or 2 (fully correct), if the question allowed partial answers, or 0 (incorrect) and 1 (correct) if the question was a dichotomous item. Grading was done according to the answer key provided by OECD (2004b). Example items can be found in Appendix A.

Students completed the test within an hour. The test was given in paper-pencil format. Overall, the test included 19 items in three different formats: multiple choice, short answer, and extended response. Multiple choice items were either standard 4-choice questions, or questions that required students to identify given responses in terms of their accuracy (i.e., true/false or correct/incorrect questions). In short answer items, students were asked to construct a short answer either in the form of a numerical answer, or a word or short phrase. Extended response

items required “more extensive writing, showing a calculation and frequently included some explanation or justification” (OECD, 2005a).

The test assessed three specific problem types, which are context-free and commonly encountered in real-life, and emphasize the process of problem solving rather than possession of domain knowledge (OECD, 2003). These problem types were *decision-making* (assed with 7 items), *system analysis and design* (assed with 7 items), and *troubleshooting* (assed with 5 items). A combination of these skills indicates the general problem solving skill of each student.

As discussed previously, teaching these problem-solving skills were an important part of the GDL intervention, and GDL tasks were aligned to provide practice with these skills. The connection between GDL tasks and PISA problem solving tasks can be seen in Table 16. The specific connection between each task and the problem solving skills involved is discussed in Chapter 3.

Table 16. *Alignment between the PISA Test and GDL Tasks*

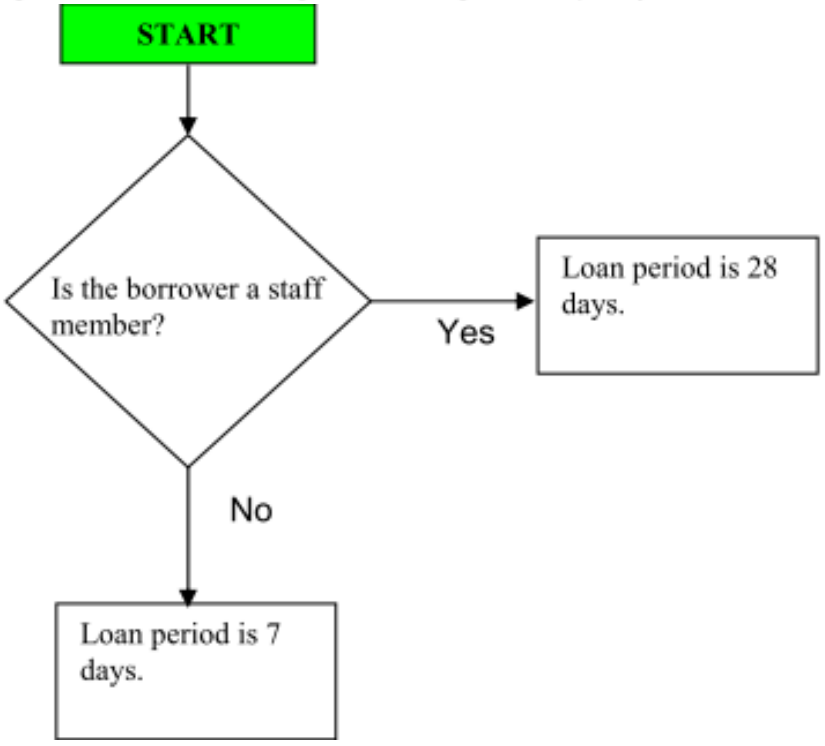
Task Category	Subtasks	Problem Solving Alignment
Game Design	• Reverse-engineer game from given GRASPS • Create flowchart	• System analysis and design • Troubleshooting • Decision making
Troubleshooting	• Find inoperable parts of a game and fix them • Create a flowchart of the game • Create GRASPS of the game	• Troubleshooting • System analysis and design • Recognize patterns • Decision making
Problem solving	• Analyze a given problem scenario • Find solution to the given problem • Understand problem patterns from data and graphs • Create simulation of the problem scenario • Create flowchart of the scenario •	• Metaskills (plan, devise solution, execute, evaluate) • Recognize patterns • System analysis and design • Troubleshooting • Decision making •
Free design	• Create flowchart of the game • Create GRASPS of the game • Troubleshoot when necessary • Make decisions based on software limitations	• System analysis and design • Troubleshooting • Decision making

Item and student response samples. System analysis and design questions "require a student to analyze a complex situation in order to understand its logic and/or to design a system that works and achieves certain goals given information about the relationships among features of the problem context" (OECD, 2003, p. 163). An example question that measured this in the test was the library system question, where the students were asked to draw the library system of an imaginary school using a flowchart. The first question (Figure 10) requires students to be able

to read the flowchart to understand the library system and give a simple answer by writing down the number in the provided space.

Figure 10. *Sample System Analysis and Design Question (PISA)*

The **John Hobson High School** library has a simple system for lending books: for staff members the loan period is 28 days, and for students the loan period is 7 days. The following is a decision tree diagram showing this simple system:



The **Greenwood High School** library has a similar, but more complicated, lending system:

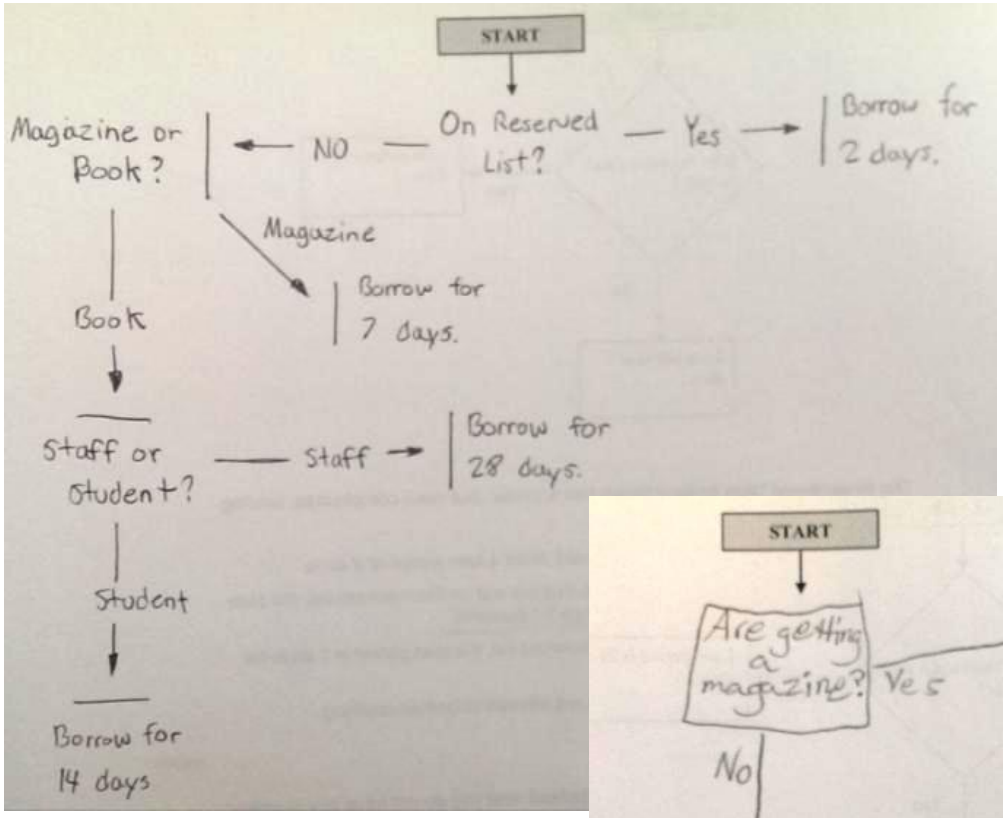
- All publications classified as "Reserved" have a loan period of 2 days.
- For books (not including magazines) that are **not** on the reserved list, the loan period is 28 days for staff, and 14 days for students.
- For magazines that are **not** on the reserved list, the loan period is 7 days for everyone.
- Persons with any overdue items are not allowed to borrow anything.

Question 1: LIBRARY SYSTEM

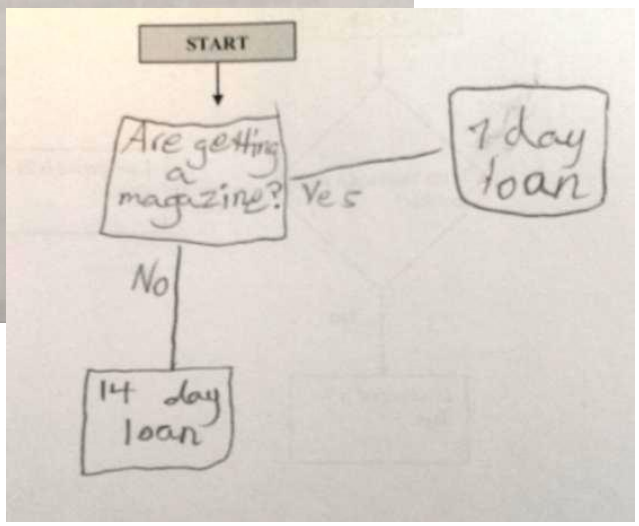
You are a student at **Greenwood High School**, and you do not have any overdue items from the library. You want to borrow a book that is **not** on the reserved list. How long can you borrow the book for?

Based on the information given about the "Greenwood High School," the second question requires students to create a flowchart for that schools' library system. Sample correct and incorrect student responses can be seen in Figure 11. The first picture represents an incorrect answer because it does not fully show the library system of the "Greenwood High School," it represents an incorrect system. In the second picture, however, we see an answer that fully depicts how the library system at the school works, covering every step of the process.

Figure 11. *Incorrect and Correct Student Answer (System Analysis and Design)*



(Correct response)



(Incorrect response)

Decision making problems "require students to understand a situation involving a number of alternatives and constraints and to make a decision that satisfies the constraints" (OECD, 2003, p. 160). An example question that measured this in the test was an item about energy needs: students were asked to calculate and decide whether a meal would be enough to satisfy the needs of a given person based on that person's daily energy needs. The first question (Figure 12) requires students to be able identify a person's energy needs by looking at data tables and deciding to which category the person belongs.

Figure 12. *Sample Decision Making Question (PISA) - 1*

This problem is about selecting the suitable food to meet the energy needs of a person in Zedland. The following table shows the recommended energy needs in kilojoules (kJ) for different people.

Age	Activity Level	Men Energy Need (kJ)	Women Energy Need (kJ)
18 to 29	Light	10660	8360
	Moderate	11080	8780
	High	14420	9820
30 to 59	Light	10450	8570
	Moderate	12120	8990
	High	14210	9790
60 and above	Light	8780	7500
	Moderate	10240	7940
	High	11910	8780

Activity Level According to Occupation:

Light:	**Moderate:**	**High:**
Indoors sales person	Teacher	Construction worker
Office worker	Outdoor salesperson	Labourer
Housewife	Nurse	Sportsperson

Question 1: Energy Needs

Mr. David Edison is a 45-year old teacher. What is his recommended daily energy need in kJ?

Answer:…….kJ.

A more complicated decision making question is the follow-up question where they read information about a person and need to go through correct calculations and make the right decision as to whether the person should or should not eat a given menu (Figure 13). A sample correct student answer can be seen in Figure 14.

Figure 13. *Sample Decision Making Question (PISA) - 2*

Jane Gibbs is a 19-year old high jumper. One evening, some of Jane's friends invite her out for dinner at a restaurant. Here is the menu:

MENU (Numbers refer to kilojoule [kJ])

Soups: Tomato Soup: 355**,** Cream of Mushroom Soup: 585
Main courses: Mexican Chicken: 960, Ginger Chicken: 795**,** Pork and Sage Kebabs: 920
Salads: Potato Salad: 750, Spinach and Apricot Salad: 335, Couscous Salad: 480
Desserts: Apple Crumble: 1380**,** Cheese Cake: 1005**,** Carrot Cake: 565,
Milk Shakes: Chocolate: 1590, Vanilla: 1470

Fixed Menu

Cost: 50 Zeds
Tomato Soup
Ginger Chicken
Carrot Cake

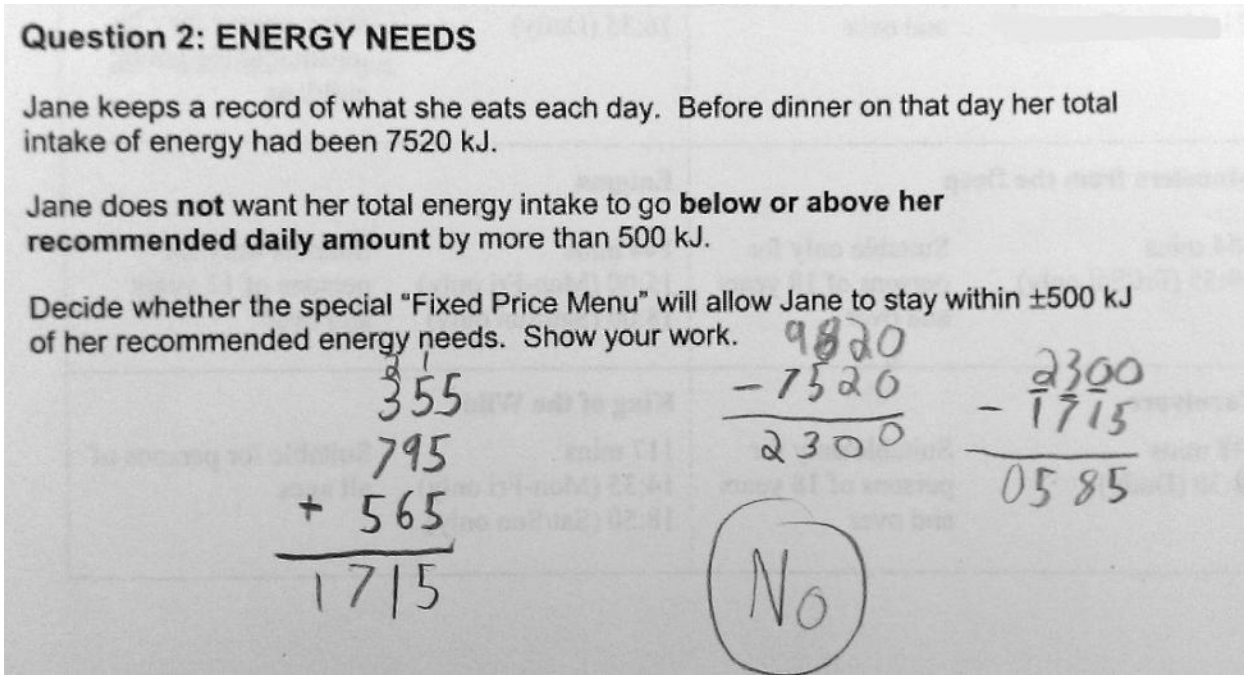
Question 2: Energy Needs

Jane keeps a record of what she eats each day. Before dinner on that day her total intake of energy had been 7520 kJ.

Jane does not want her total energy intake to go below or above her recommended daily amount by more than 500 kJ.

Decide whether the special "Fixed Menu" will allow Jane to stay within ±500 kJ of her recommended energy needs. Show your work.

Figure 14. *Sample Correct Student Response (Decision Making)*



Finally, troubleshooting problems "require a student to comprehend the main features of a system and to diagnose a faulty or under-performing feature of the system or mechanism" (OECD, 2003, p. 168). An example question that measured this in the test was the irrigation item: the students were asked to analyze an irrigation system and find a faulty gate that prevents the water flow (Figure 15). Giving the correct answer requires students to attend the details of the system, to understand it, and find out what is causing the system to fail.

Figure 15. *Sample Troubleshooting Question (PISA)*

Below is a diagram of a system of irrigation channels for watering sections of crops. The gates A to H can be opened and closed to let the water go where it is needed. When a gate is closed no water can pass through it.

This is a problem about finding a gate which is stuck closed, preventing water from flowing through the system of channels.

Michael notices that the water is not always going where it is supposed to.

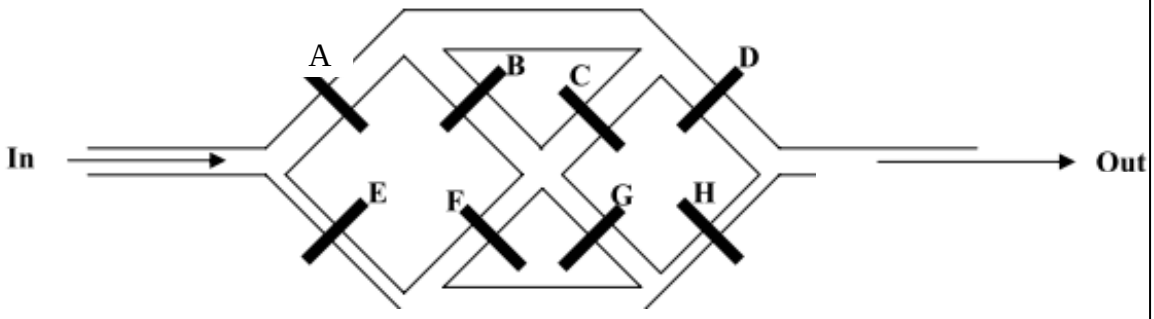
He thinks that one of the gates is stuck closed, so that when it is switched to "open", it does not open.

Question 1: IRRIGATION

Michael uses the settings given in Table 1 to test the gates.

A	B	C	D	E	F	G	H
Open	Closed	Open	Open	Closed	Open	Closed	Open

With the gate settings as given in Table 1, **on the diagram below** draw all the possible paths for the flow of water. Assume that all gates are working according to the settings.



PISA assessment was analyzed using item-response-theory based (IRT) methods. In IRT analyses each individual item in the test has data on its difficulty level, which, in a further IRT analyses is used to calculate each students' problem solving proficiency levels (See Appendix B for details on calculating student proficiencies). Calculated according to IRT procedures, each

student's problem solving ability was calculated on a scale ranging from -4 to 4. This scale can roughly be interpreted as a student's probability of answering all the items at a test correctly. Specifically, a student with an ability level of 0 on a specific question has 50% answering the question correctly.

Motivation variables. To measure students' situational and emerging individual interest and utility value perceptions, self-report surveys was used (see Appendix C). Participants responded to all motivation items on a 100-point scale from 0 (*strongly disagree*) to 100 (*strongly agree*). A summary of reliability statistics for each scale in both pre- and post-implementations can be found in Table 17.

Table 17. *Motivation Scale Reliabilities by Domain and Time*

Domains	Scales	Item *n*	Pre *a*	Post *a*
Game design				
	Utility value	7	.95	.90
	Situational interest	5	.83	.78
	Emerging individual Interest	4	N/A	.77
Programming				
	Utility value	7	.95	.91
	Situational interest	5	.82	.84
	Emerging individual Interest	4	N/A	.79
Problem Solving				
	Utility value	7	.95	.95
	Situational interest	5	.92	.94
	Emerging individual Interest	4	N/A	.85

Perceived utility value. Participants' perceptions of utility value (e.g., "I believe programming could be of some value to me.") of game design, programming, and problem-solving activities was measured by a 21-item scale (7 items for each activity). The survey was adapted from Hulleman et al. (2010), where the reliability of this scale was acceptable (ranging from α=.78 to α=.88).

Situational interest. Participants' situational interest (e.g., "I think learning programming at this camp is interesting.") in the game design, programming, and problem-solving activities was measured by a 15-item scale (5 items for each). The items were adapted from an early research by Hulleman et al. (2010), where the reliability of this scale was high (α=.89).

Emerging individual interest. Participants' emerging individual interest in the game design, programming, and problem-solving activities was measured by a 12-item scale (e.g., "My experience at this camp makes me want to take more programming classes," "I am not really interested in using programming in my future career"). The items were adapted from a related study by Hulleman and Harackiewicz (2009), where the reliability of this scale was high (α=.84).

CHAPTER 5

Results

Changes in Problem-Solving Skills

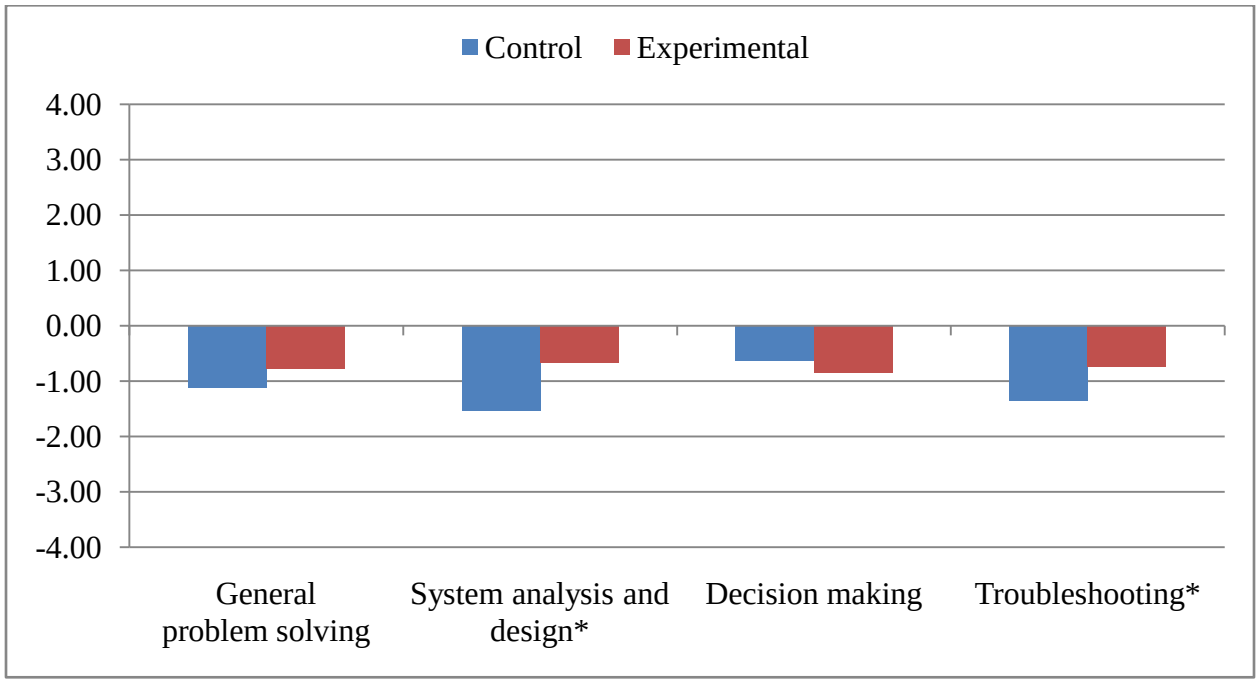
Research question # 1. To answer the first research question, namely to understand if students attending the GDL courses showed significantly higher gains in general and specific problem solving skills (i.e., decision-making, system analysis and design and troubleshooting) compared to the control group, I used two tests.

First, to understand if the performances of the two groups were similar at the pre-test, I ran a one-way multivariate analysis of variance (MANOVA), having two levels of group (experimental vs. control) with general problem solving, decision-making, system analysis and design, and troubleshooting skills at pre-test as dependent variables. This analysis is an important first step to understand if the groups were similar at the pre-test. Because the groups were not chosen randomly, i.e. a quasi-experimental research design, potential significant group differences (e.g., history, maturation, testing) can confound the results and threat the internal validity of the research. Understanding the difference of the groups in the pre-test can account for these types of threats to the study's internal validity.

The multivariate between-subjects omnibus for group (i.e., experimental vs. control) was significant (Wilks's Λ = .771), F (4, 68) = 5.057, p = .001, η^2 =.229. Specifically, there were significant differences between the control and experimental groups at the pre-test on system analysis and design, F (1, 71) = 9.139, p = .003, η^2 =.114; and troubleshooting, F (1, 71) = 4.591, p = .036, η^2=.061, favoring the experimental group. Differences between the groups at the pre-test was not statistically significant for the measures of general problem solving, F (1, 71) =

2.053, p = .156, η^2 =.028, and decision-making skills, F (1, 71) = .647, p = .424, η^2=.009. The results indicate that the two groups were not statistically different in terms of their general problem-solving skills at the pre-test. As it can be seen in Figure 16, the students at the experimental sites had higher initial levels of problem solving. In IRT, the scale from -4 to 4 indicates the students' ability level (i.e., theta) or the likelihood of getting a question correct. The higher the student's theta, the more likely the student can get the question correct. Zero, in this scale refers to 50% chance.

Figure 16. *Problem Solving Abilities for Control and Experimental Groups at the Pretest*



Next, in order to understand if the two groups differed in terms of gains they made in general and specific problem solving skills over time, I ran a repeated-measures multivariate analysis of variance test (RM-MANOVA), having two levels of time (pre vs. post) as within-subjects factors, and two levels of group (control vs. experimental) as between-subjects factors. The multivariate omnibus for group was significant (Wilks's Λ = .64), F (4, 68) = 9.564, p<.001,

η^2=.36, as were within-subjects test for time (Wilks's Λ = .783), F (4, 68) = 4.722, p = .262, η^2p=.06, and Time X Group interaction (Wilks's Λ = .659), F (4, 68) = 8.793, p<.001, η^2=.341.

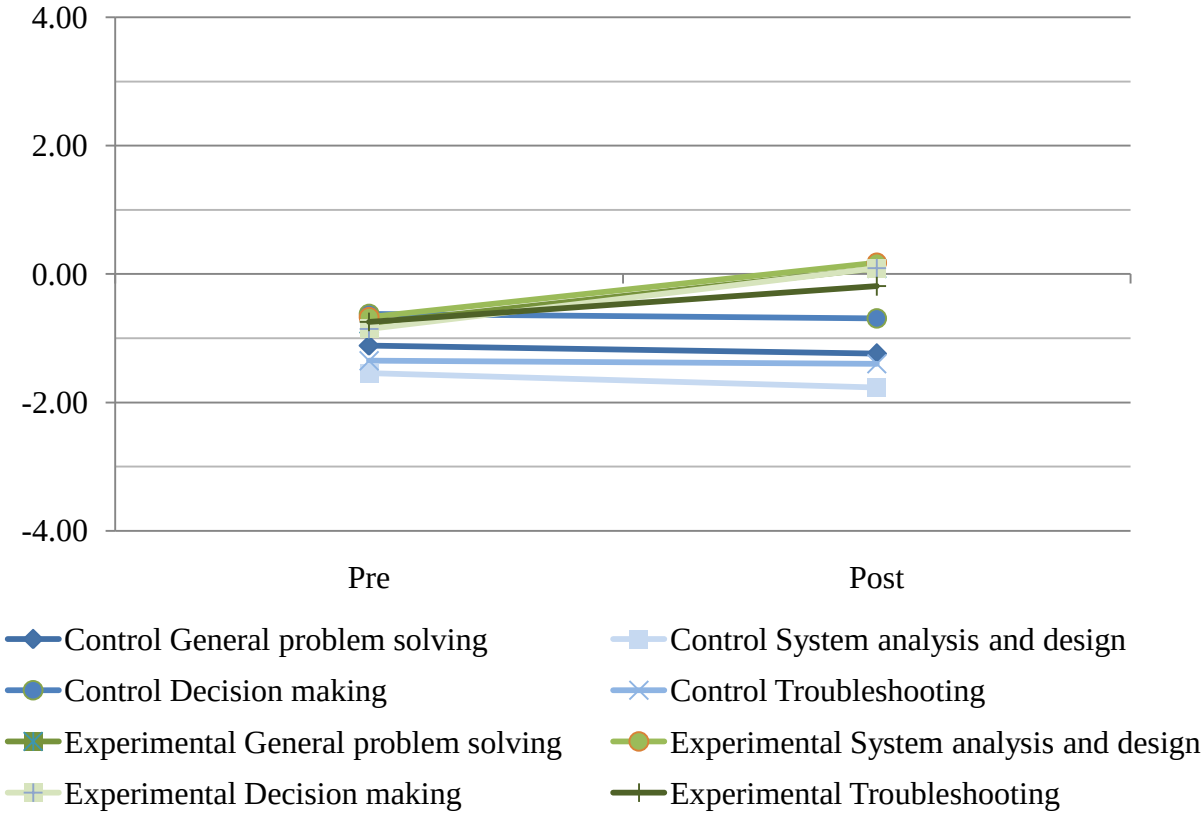
Follow-up between-subjects tests indicated that experimental and control groups significantly differed in general problem solving, F (1, 71) = 14.368, p<.001, η^2=.17; system analysis and design, F (1, 71) = 31.696, p<.001, η^2=.31; and troubleshooting, F (1, 71) = 14.699, p<.001, η^2=.17, but not for decision making, F (1, 71) = 1.18, p =.281, η^2=.015. Time x Group interaction was significant for all tests: general problem solving, F(1, 71) = 35.131, p<.001, η^2=.29; system analysis and design, F(1, 71) = 18.781, p<.001, η^2=.195; decision-making, F(1, 71) = 15.93, p = .281, η^2=.161, and troubleshooting, F(1, 71) = 4.745, p = .033, η^2=.051, favoring the experimental group in all of the tests. This result indicates that experimental sites as a whole, compared to the control group, showed statistically significant gains in general problem solving, system analysis and design, and troubleshooting, while the changes for control group were not statistically significant. Table 18 and Figure 17 summarize the results for each domain. These results favor the experimental groups.

Table 18. *Descriptive Statistics for General and Specific Problem Solving (Control vs. Experimental)*

		General PS		SYS		DM		TS	
		Pre	Post	Pre	Post	Pre	Post	Pre	Post
Experimental	*M*	-0.77	0.09	-0.67	0.17	-0.85	0.09	-0.74	-0.19
	SD	1.08	0.91	1.17	1.04	1.34	1.06	1.21	1.06
Control	*M*	-1.12	-1.24	-1.54	-1.77	-0.62	-0.69	-1.35	-1.4
	SD	0.63	0.98	1.14	1.16	0.62	1.2	0.97	1.08

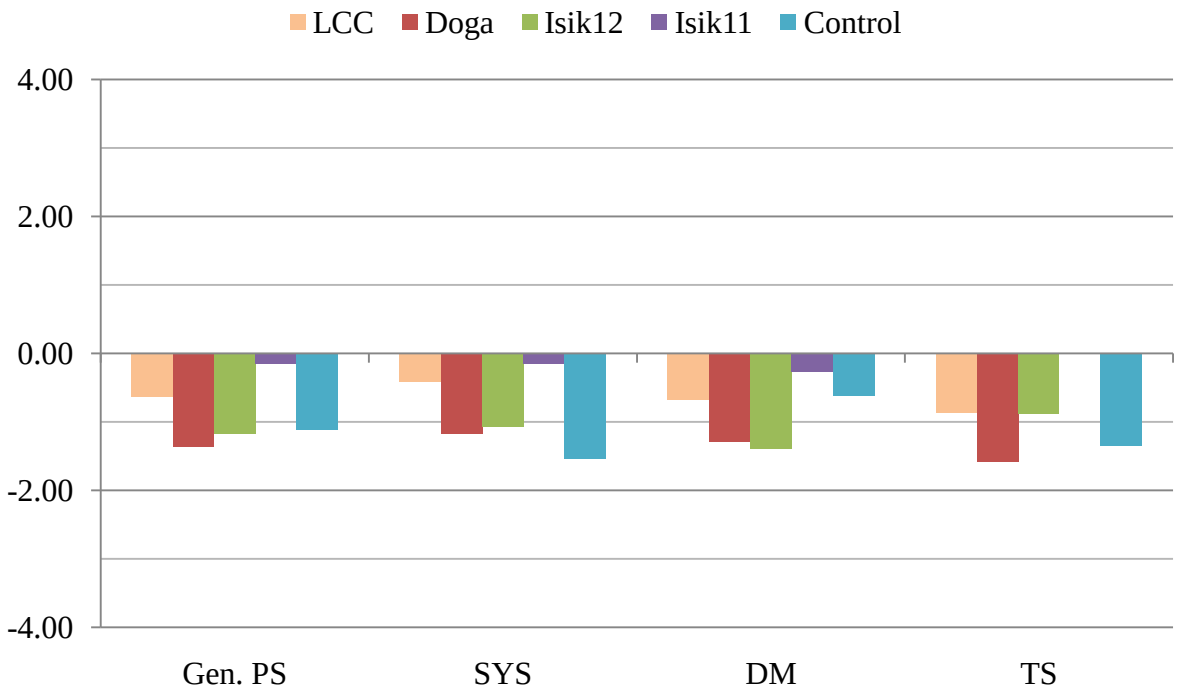
**General PS = General Problem Solving, SYS = System analysis and design, DM = Decision making, TS = Troubleshooting*

Figure 17. *General and Specific Problem Solving Skill Changes for Control and the Experimental Groups*



Research question # 2. Next to answer whether there were differences between sites in the gains that they showed in their problem-solving skills, I ran two tests. First, in order to see if the sites had any differences at the pre-test in terms of general and specific problem-solving skills, I conducted a MANOVA having five levels of site (Isik11, Isik12, Doga, LCC, and control) as between-subjects factor with general problem-solving, decision-making, system analysis and design, and troubleshooting skills at the pre-test as dependent variables. The multivariate omnibus for site was significant (Wilks's Λ = .549), F (16, 199.216) = 2.699, p = .001, η^2p=.139, indicating that the groups differed in their initial skills in terms of general and specific problem solving skills (Figure 18).

Figure 18. *Problem Solving Skill Differences at the Pretest for Different Sites*



**General PS = General Problem Solving, SYS = System analysis and design, DM = Decision making, TS = Troubleshooting*

Between-subjects tests indicated that there were significant group differences for all tests: general problem solving, $F(4, 68) = 5.027$, $p = .001$, $\eta^2 = .23$; system analysis and design, $F(4, 68) = 4.511$, $p = .003$, $\eta^2 = .21$; decision making, $F(4, 68) = 2.632$, $p = .042$, $\eta^2 = .134$; and troubleshooting, $F(4, 68) = 6.017$, $p < .001$, $\eta^2 = .261$.

To understand specific group differences I conducted follow-up tests with Bonferroni adjustment. In terms of general problem solving skills, there were significant differences between Isik11 group, and Doga ($p = .003$), Isik12 ($p = .032$), and the Control groups ($p = .007$), where Isik11 group's initial performance was significantly higher than the others. In system analysis and design, there was a significant difference between Isik11 and the control group ($p =$

.002), favoring the Isik11 group. In decision-making, there were not any significant differences among the groups. Finally, in terms of troubleshooting, there were significant differences between Isik11 and Doga (p = .001) and Control sites (p = .001), again favoring the Isik11 site's initial performance. A summary of the means and standard deviations can be found in Table 19.

Table 19. *Descriptive Statistics for Problem Solving Tests for Each Site*

Dependent variables	LCC		Doga		Isik12		Isik11		Control	
	M	*SD*	*M*	*SD*	*M*	*SD*	*M*	*SD*	*M*	*SD*
General problem solving										
Pretest	-0.64	0.87	-1.37	0.87	-1.17	0.87	-0.15	0.87	-1.12	0.63
Posttest	0.20	0.88	-0.13	0.88	-0.51	0.88	0.57	0.88	-1.24	0.98
System analysis and design										
Pretest	-0.42	1.12	-1.17	1.12	-1.08	1.12	-0.16	1.12	-1.54	1.14
Posttest	0.25	1.05	-0.12	1.05	-0.28	1.05	0.63	1.05	-1.77	1.16
Decision-making										
Pretest	-0.68	1.11	-1.29	1.11	-1.40	1.11	-0.27	1.11	-0.62	0.62
Posttest	0.23	1.08	0.04	1.08	-0.61	1.08	0.50	1.08	-0.69	1.20
Troubleshooting										
Pretest	-0.87	1.03	-1.59	1.03	-0.89	1.03	0.00	1.03	-1.35	0.97
Posttest	-0.07	1.00	-0.47	1.00	-0.87	1.00	0.38	1.00	-1.40	1.08

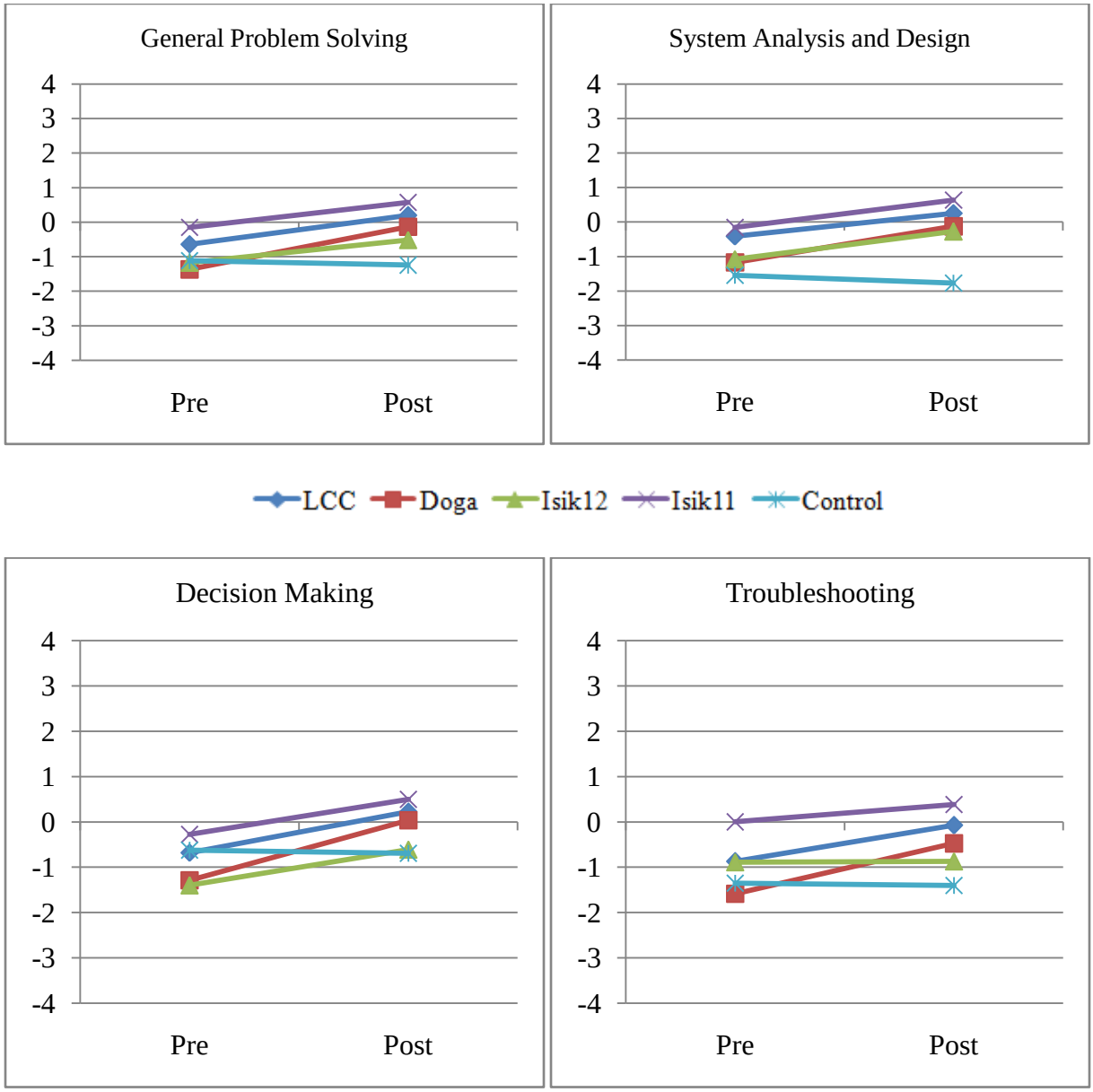
Next, to understand the differences in the sites in terms of the changes that they showed in general and specific problem-solving skills, I conducted an RM-MANOVA, having two levels of time (pre vs. post) and five levels of site (Isik11, Isik12, Doga, LCC and Control) as between-subjects factor. The multivariate omnibus for site was significant, (Wilks's Λ = .467), F(16, 199.216) = 3.521, p < .001, η^2 =.537, so was the main effect of time (Wilks's Λ = .511), F(4, 65) = 15.521, p<.001, η^2 =.489. The interaction between Time X Site was also significant (Wilks's Λ

= .57), $F(16, 199.216) = 2.518$, $p = .002$, $\eta^2 = .43$. This indicates that the there were differences in the gains the sites showed.

Follow-up between-subjects tests for site indicated that for all problem-solving tests there was a significant impact of the site: general problem solving, $F(4, 68) = 8.224$, $p < .001$, $\eta^2 = .33$; system analysis and design, $F(4, 68) = 10.933$, $p < .001$, $\eta^2 = .39$; decision making, $F(4, 65) = 2.912$, $p = .028$, $\eta^2 = .15$, and finally for troubleshooting, $F(4, 65) = 0.155$, $p < .001$, $\eta^2 = .35$.

Post-hoc tests with Bonferroni adjustment indicated that for general problem-solving skills, the gains that Isik11 showed was significantly larger than Doga ($p = .018$), Isik12 ($p = .012$), and Control sites ($p < .001$). The gains for LCC and Isik11 groups did not significantly differ. As for system analysis and design, there were significant differences between the Control group and LCC ($p = .003$), and Isik11 ($p < .001$). Control group showed significantly smaller gains than these two sites, but not Doga and Isik12 sites. In terms of decision-making, the only significant difference was between Isik11 and Isik12 ($p = .035$), favoring the gains Isik11 showed. Finally, in troubleshooting, Isik11 group showed significantly larger gains than Doga ($p = .002$), Isik12 ($p = .018$), and Control ($p < .001$), but not LCC site. Descriptive statistics can be found in Table 16 (above). Pre-Post changes at each site by problem type can be seen in Figure 19.

Figure 19. *Gains in Problem Solving in Each Problem Solving Test for Each Site*



Changes in Motivation

Before going forward with the analyses in this section, I ran Kolmogorov-Smirnova and Shapiro-Wilk tests to understand if the motivation data met the assumptions for inferential statistical analyses. As it can be seen in Table 20, the results of both tests indicated that all factors in both the pre- and the post-surveys were significantly non-normal and violated the

assumption of normality. Q-Q plots also confirmed these results, indicating that observations deviated substantially from the normality plot.

To understand the reasons for this non-normality, I also investigated descriptive statistics. Descriptive statistics showed that there was a ceiling effect (i.e., very high medians) for all factors, where the students picked very high initial and equally high post values (Table 21). In other words, the assessments could not detect whether or not students' motivation was likely to change because it was so high in the beginning and the assessments could not measure much change.

Due to this violation and the ceiling effects, I could not conduct any further analyses and answer research questions regarding changes in motivation, impact of the relevance intervention, and the relationship between motivational and cognitive constructs.

Table 20. *Normality Assumption Statistics for Motivation Factors*

			Kolmogorov-Smirnova			Shapiro-Wilk		
		Pre	Statistic	df	Sig.	Statistic	df	Sig.
Pre	Utility Value	Game design	0.248	16	0.009	0.850	16	0.013
		Programming	0.252	16	0.008	0.798	16	0.003
		Problem solving	0.229	16	0.025	0.770	16	0.001
	Situational Interest	Game design	0.235	16	0.018	0.817	16	0.005
		Programming	0.289	16	0.001	0.668	16	0.000
		Problem solving	0.230	16	0.023	0.838	16	0.009
Post	Utility Value	Game design	0.328	16	0.000	0.765	16	0.001
		Programming	0.319	16	0.000	0.673	16	0.000
		Problem solving	0.299	16	0.000	0.715	16	0.000
	Situational Interest	Game design	0.378	16	0.000	0.670	16	0.000
		Programming	0.312	16	0.000	0.666	16	0.000
		Problem solving	0.304	16	0.000	0.666	16	0.000
	Emerging individual interest	Game design	0.324	16	0.000	0.720	16	0.000
		Programming	0.337	16	0.000	0.684	16	0.000
		Problem solving	0.214	16	0.048	0.809	16	0.004

Table 21. *Descriptive Statistics for Motivation Factors*

									Percentiles		
			M	*Mdn*	*SD*	Range	Min.	Max.	25	50	75
Pre	Utility value	GD	85.1	91.5	15.5	45.4	54.6	100	68.8	91.5	98.5
		Prog.	89.4	95.4	12.3	31.4	68.6	100	76.6	95.4	100
		PS.	89.9	95.5	13.6	49.1	50.9	100	84.9	95.5	100
	Situational interest	GD	94.2	97.0	6.4	17.2	82.8	100	87.7	97.0	100
		Prog.	95.5	99.7	7.4	21.6	78.4	100	91.5	99.7	100
		PS.	90.5	95.5	10.5	30.0	70.0	100	83.6	95.5	99.8
Post	Utility value	GD	90.9	98.3	11.8	32.9	67.1	100	80.3	98.3	99.5
		Prog.	92.3	98.2	12.8	41.6	58.4	100	87.8	98.2	100
		PS.	90.6	98.4	14.5	46.4	53.6	100	86.1	98.4	100
	Situational interest	GD	95.7	98.5	7.2	24.0	76.0	100	92.5	98.5	100
		Prog.	95.4	100	7.8	24.6	75.4	100	91.7	100.0	100
		PS.	90.6	99.3	15.9	52.0	48.0	100	88.2	99.3	100
	Emerging individual interest	GD	91.6	97.8	11.3	27.5	72.5	100	78.4	97.8	100
		Prog.	93.7	99.3	9.9	27.8	72.3	100	86.6	99.3	100
		PS.	85.7	93.8	17.3	50.0	50.0	100	68.7	93.8	100

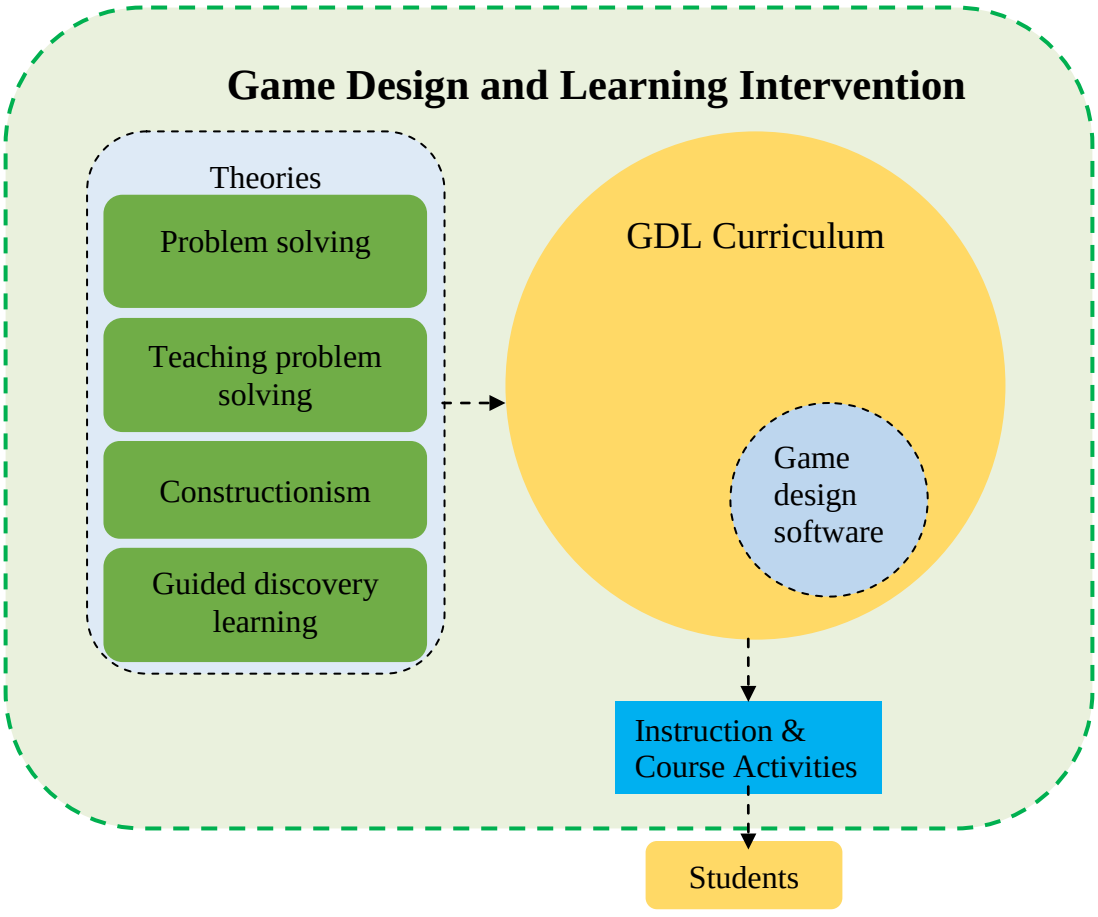
GD = Game design, Prog. = Programming, PS = Problem solving

CHAPTER 6

Discussion

GDL intervention was composed of series of activities that offered young children a chance to design games, learn computer programming, and practice their problem-solving skills. Although the courses were offered at varying different lengths and at different sites, the results of the study indicated that the students showed significant improvement in their problem-solving skills as compared to a control group who did not attend the GDL courses. As described in Chapter 3, the GDL intervention was multi-faceted (Figure 20), composed of different activity types (informed by previous research and theory), and each aligned to teach students game design, programming and problem solving skills.

Figure 20. *Multi-faceted Structure of the GDL Intervention*



The primary purpose of this study was to investigate the impact of learning game design on middle school students' general and specific problem-solving skills (system analysis and design, decision making, troubleshooting). The results indicated that the students who received GDL intervention performed significantly better than the control group students who did not attend the GDL courses. In other words, through the GDL intervention (i.e., tasks on game design, troubleshooting and free design) students were able to show development in problem-solving skills that are important for their future lives and careers.

In addition, in this study I also aimed to show the impact of learning game design on students' utility value and interests in game design, programming, and problem solving. The motivation data, however, severely violated the assumptions of normality, making it impossible to answer the research questions regarding changes in students' motivation. Despite this, however, motivation data pointed to two important aspects about the participants that can be positively interpreted: (a) they were highly motivated for game design, programming, and problem solving, (b) their post survey results point to the fact that they remained equally, if not slightly more, motivated for these tasks. Given the limitations of quasi-experimental research, this study's positive outcomes in problem solving should be interpreted with caution.

First the positive outcomes cannot solely be attributed to the GDL intervention, as there can be external and internal threats to validity. One of the main threats to internal validity in such a design as used in this study is the non-randomized selection of the participants. As it was indicated in the results chapter, there were some significant differences in the initial levels (i.e., pre-test) among the sites (experimental vs. control), and within experimental sites. Also, there were differences between the gains that different sites showed in general and specific problem solving skills. The initial skill level differences might have led them to benefit differently from

the GDL intervention, or to show different levels of gain over time. This particular unintended selection-bias, therefore, might have threatened the internal validity of the findings, and might be responsible for some of the gains in the experimental sites (Campbell & Stanley, 1963).

Second, the students participating at the GDL courses were highly motivated in game design, programming, and problem solving, as is reflected in the motivation survey results. My casual observations as the lead instructor at the GDL courses also support this argument. For example, after each GDL course a common question was when the next course would be and if we could offer a higher-level course. In terms of motivation and interest, this potentially puts this study's sample at a different level from the rest of the population. Therefore, results should be interpreted within these limits. More detail about the limitations can be found in later in this chapter, in the "Limitations" section.

Changes in Problem Solving Skills

Discussion for general problem solving skills. The results of this study showed that, compared to the control group, the students who attended the GDL courses showed significantly higher gains in their general and specific problem-solving skills as measured by the 2003 PISA Problem Solving Assessment. This indicates that the GDL curriculum helped students make progress in their specific (i.e., system analysis and design, decision making, and troubleshooting) and general problem-solving skills —as measured by the combination of the specific problem solving skills— over time. To reiterate, system analysis and design refers to students' ability to identify components of a system and build new systems, satisfying many interrelated variables. Decision-making refers to making the best decision (i.e., cost-effective) based on the limitations in a given case. Finally, troubleshooting tasks involve finding and fixing the inoperable part of a system that is otherwise functional.

The results also indicated that, despite the differences in the initial levels of problem solving, the students at the different experimental sites showed significant gains in their problem-solving skills. This impact across sites indicates the robustness of the findings and the GDL curriculum. In other words, students at all sites showed significant gains in all problem-solving skills measured, although the results also pointed to significant gain differences among some sites. Based on these results, GDL courses seem to be an effective way of teaching problem-solving skills to middle school students.

One possible explanation for these positive outcomes of GDL courses can be the effectiveness of the curriculum, more specifically the specific attention paid to establish game design tasks as an effective method of teaching problem solving. As it was explained in the early chapters, GDL curriculum was built on theories of teaching problem solving and these methods were implemented throughout the intervention via various activities. These activities were great platforms for students to learn game design and programming basics, but more importantly to learn metacognitive skills underlying problem solving, in addition to gaining practice and experience in system analysis and design, decision making, and troubleshooting tasks.

Research on teaching of problem solving provides us with a wide spectrum of methods, and these methods vary in terms of their effectiveness. Interested readers can find an exhaustive summary of the methods in the two comprehensive literature reviews by Mayer and Wittrock (1996, 2006). The results of the current study showed that by carefully establishing the connection between problem-solving processes and game design tasks, and embedding proven methods of teaching problem solving into the game design curricula, game design can be put forward as one of the newer methods of teaching problem solving.

The GDL curriculum was based on some well-established methods of teaching problem solving: teaching basic skills (automaticity & constraint-removal), teaching skills directly, structure-based methods, generative methods and teaching by analogy. These specific methods were achieved through four specific tasks: (a) game design, (b) problem solving, (c) troubleshooting, and (d) free design. The purpose of each task and their alignment with problem-solving skills can be seen in Table 22.

Game design tasks were offered during the first half of the GDL courses with aim of helping students master game design and programming basics as well as master the game design software during this time. Underlying theory in this design was to implement automaticity and constraint-removal methods. Through these methods, the aim was to lessen students' cognitive load by automatizing basic tasks and leaving more room for tasks that require high-level skills (Mayer & Wittrock, 1996). At GDL, during the first half of the course, students mastered the game design process so that they were able to devote more thinking into working on problem solving scenarios.

Table 22. *GDL Activities and Their Alignment with Problem Solving Skills*

Phase	Category	Purpose	Subtasks	Problem solving
#1	Game Design (Guided)	• Game design • Programming • Microsoft Kodu	• Reverse-engineer game from given GRASPS • Create flowchart	• System analysis and design • Troubleshooting • Decision making
#2	Problem solving	• Problem solving skills • Familiarize with complex problems • *Programming* • *Microsoft Kodu*	• Analyze a given problem scenario • Find solution to the given problem • Understand problem patterns from data and graphs • Create simulation of the problem scenario • Create flowchart of the scenario	• Metaskills (plan, devise solution, execute, evaluate) • Recognize patterns • System analysis and design • Troubleshooting • Decision making
#2	Troubleshooting	• Troubleshooting • Familiarize with complex problems • *Programming* • *Microsoft Kodu*	• Find inoperable parts of a game and fix them • Create a flowchart of the game • Create GRASPS of the game	• Troubleshooting • System analysis and design • Recognize patterns • Decision making
#2	Free design	• Game design • *Programming* • *Microsoft Kodu*	• Create flowchart of the game • Create GRASPS of the game • Troubleshoot when necessary • Make decisions based on software limitations	• System analysis and design • Troubleshooting • Decision making

Game design activities were also ideal environments for students to practice their system analysis and design, troubleshooting, and decision-making skills. More specifically, the flowcharts the students created before and after they created their games can be considered one of the ways that the students practice their system analysis and design skills. This can be because the flowcharts provide students with a visual context to see games as a system and build confidence in their system analysis and design skills.

Troubleshooting is naturally built in to the game design process and students practice their troubleshooting skills during each and every game design task. For example, during the courses it is very usual to see students who try to implement something they see in the games they play (e.g., teleportation from one gate to another), and would like to implement it in Kodu. Because Kodu does not offer direct ways to do this, students often need to be very creative in repurposing lines of code. During this process, and many similar instances, students often experience a roadblock. The students themselves often reveal the solutions to these roadblocks by going back and analyzing each line of code until they find the one that is misplaced. Similarly, during this process, students often face moments where they need to make a decision in order to how to implement a certain element in their game. They need to evaluate constraints (e.g., the computer processing power, limitations of the software, limitation of codes, limitations posed by game structure) and make the best decision to create the best game that they can.

During the second phase of GDL, through *problem-solving tasks*, the students are offered ways of practicing problem-solving skills by working on solving real-life problems, and then recreating problem scenarios in Kodu. During these tasks, multiple methods of teaching problem solving were used, and it can be speculated that these were important in helping students improve their problem-solving skills. The method of teaching skills directly was used in the form

of showing students the important steps in tackling every problem: understand, plan, execute, evaluate (Polya, 1957). Through guidance of the instructors, the students got chances to work on seemingly complex problems, and practiced important metacognitive skills underlying problem solving.

Another method that was used during problem solving tasks was use of analogies. Through the use of analogies, instructors aimed to show students that, although problems might differ on the surface, they also might share similarities. By providing analogous problems to students during GDL, the students had a chance to practice important skills to solve analogies, such as seeing patterns. The problem-solving tasks offered during the second phase of GDL were also very good examples of structure-based methods and generative methods. Structure-based methods involve usage of objects (physical or virtual) to help solve problems. At GDL, the re-creation of games in Kodu can be considered as an example of this method, where simulations created in Kodu can be considered as the "objects" that students manipulate and think with. Similarly, generative methods involves students relating students own experiences with the problem, and re-creating problem scenarios in Kodu gave students chances to create problem scenarios with the same underlying principle, but within their own imaginative worlds.

Troubleshooting activities, as the name suggests, were specifically designed and offered to provide extra practice and guidance to students in their troubleshooting skills. By fixing a broken game, and creating its flowchart after, students had a chance to practice both their troubleshooting and system analysis and design skills. Decision making was also embedded in this process, where students needed to make informed decisions about where to look for the source of the problems, and which steps they needed to take in order to completely recover the broken game.

Finally, *free game design activities* provided students with good context to both practice their game design skills, and also use system analysis and design (i.e., create flowchart of their games), decision-making, and troubleshooting skills. This task was also important because of the personal attachment the students had with their products. Because of this personal attachment, student engagement in these tasks was at high levels.

These instructional techniques are effective methods of teaching problem solving (Mayer & Wittrock, 2006). From early research on programming, it is also known that instructional methods used in teaching programming can lead to differences in learning outcomes. For example, in their research, Lehrer et al. (1999) found that reflective methods of teaching LOGO (e.g., students wrote summaries of their programming experiences) was more effective than inquiry teaching (e.g., teacher-led questioning, predicting and assisting). In the current research, multiple instructional techniques were used.

Although results indicated that the instructional intervention (GDL courses) worked as a whole, it is very hard to know which instructional technique or teaching strategy contributed to the results, and to what extent. At this point it is hard to understand the impact of specific methods and their unique affordances. Important questions remain unanswered and need to be scrutinized further in future research. More specifically, we do not know how the individual instructional methods might have contributed individually, and if the methods or tasks are additive, interactive, or independent. Similarly, we do not know if the instructional methods and GDL tasks are all equally necessary or if some are more important than others. Equally relevant here is the dosage and duration of the intervention. The specific impact of length was beyond the scope of this research. The GDL courses were offered at different lengths (15-40 hours) and the results indicated experimental groups showed significantly larger gains than the control group.

The activity structure was kept the same for each site, and at the sites where more time was available, more *free design tasks* were offered to keep the activity structure comparable. It might be speculated here that there might be minimal exposure length (i.e., the minimal length for GDL), but more importantly the specific emphasis and impact of each different task needs further scrutiny.

Discussion for specific problem-solving skills. Another important result of this research was that, in addition to the general problem solving skills, the intervention resulted in significant gains in specific problem-solving skills, system analysis and design, decision-making, troubleshooting skills. These skills are considered to be very important problem-solving domains that individuals face with in their everyday lives (Jonassen, 2000; OECD, 2004a, 2012). The gains in these specific skills indicate that during game design process they can be practiced, and game design can be a suitable context to teach these specific problem-solving skills.

System analysis and design involves understanding "the complex relationships among a number of interdependent variables, identify their crucial features, create or apply a given representation, and design a system so that certain goals are achieved" (OECD, 2004, p. 39). Problems of this kind, namely design problems, are considered to be the most complex and ill-structured problem types (Jonassen, 2000). Games are systems, and design of these systems by students could have contributed to the improvement they showed in system analysis and design. The improvement in system analysis and design points out that game design can be a viable platform for teaching students' system analysis and design skills. Especially through tasks such as asking students create flowcharts of their games; this specific skill can be placed into a meaningful and engaging context.

Decision-making problems require students to make the best choice out of existing rival alternatives (Jonassen, 2000; OECD, 2004a). In the game design process, students are often faced with situations where they need to make a choice to fit best into their design-in-progress. These engaging decision-making situations come naturally out of the game design process, and due to students' personal involvement with the projects, they spend a considerable amount of time to make the best decision given the constraints of the game design software and the goals of their games. The results of this study also indicate that this process is beneficial in developing students' skills as good decision-makers.

Finally, troubleshooting is one of the most common types of problems people face in their everyday lives, which involves diagnosing a faulty mechanism in an otherwise working structure (Jonassen, 2000; OECD, 2004a). Throughout GDL courses, one of the activities that came naturally out of engaging in game design and programming process was impromptu troubleshooting by students (Guzdial, 2004). The results, again, support that through design and opportunities to troubleshoot their own games, the students improve in their troubleshooting skills. As mentioned before, representing systems through flow diagrams is very important both problem representation and also identifying faulty systems for troubleshooting (Jonassen, 2000), and the students attending GDL courses had plenty of opportunities to do so.

Discussion for site differences. Although the students at the experimental sites, compared to the control group, showed significant gains in their problem solving skills over time, statistical tests also indicated that there were site differences at the pre-test, and in terms of the overall gains. This might indicate that different sites benefitted from the GDL courses at different levels, although they each showed significant gains from pre to post. Early research with LOGO (Mayer & Fay, 1987; Palumbo, 1990) showed that learning programming is

positively impacted by students' initial levels cognitive abilities and programming knowledge. This can be a possible explanation for the differences in gains for different sites. In other words, starting at different cognitive levels, the sites with higher initial levels had higher post-test results.

Looking at student or site characteristics was beyond the scope of this study, and should be studied further. Specifically, the prerequisites (e.g., programming knowledge, basic cognitive skills, etc.) of game design and problem solving should be identified, as well as the specific student characteristics which cause them to be more successful at these tasks should be answered in future research.

Discussion summary for problem solving. Recent conceptual claims on the connection between game design and thinking skills (i.e., computational thinking) have been, unfortunately, far from providing empirical support for these arguments, pointing to a lack of both research and conceptual agreement. Through early research on programming, we have reason to believe that such a connection between game design and thinking skills is possible (Nastasi et al., 1990; Reed & Palumbo, 1992; Suomala & Alajaaski, 2002; Ziegler & Terry, 1992). The current research, therefore, is the first study in the recent literature to provide empirical support for the possible connection between learning game design in a curriculum based on teaching problem solving and problem-solving skills.

As stated previously, however, the results of this study need to be interpreted within the context of GDL curriculum and activities offered within the GDL intervention, as early research has established that simple exposure to computer software does not lead to improvement in cognitive skills (Mayer & Fay, 1987; Pea & Kurland, 1984). As depicted in this study, successful connection of game design and thinking skills involves specific attention in the creation of

curricula and activities to best make use of theories of problem solving instruction and theories of learning. It is only through this careful attention that the connection between game design and problem solving can be established and students' abilities in these domains can be scaffolded.

Motivational Impacts of the GDL Intervention

Another purpose of this study was to understand unique changes in students' interest and value in game design, programming, and problem solving, and how these were affected by the interventions in this study. Due to the problems with data, however, the analyses related to these questions could not be conducted and left for future research.

To reiterate, the questions related to changes in motivation in this study were:

- changes in students' motivational perceptions (utility value, situational interest and emerging individual interest) for game design, programming, and problem solving.
- the relationship between utility value of game design, programming and problem solving, and the emerging individual interest in these domains.
- impacts of a relevance intervention on both cognitive and motivational outcomes
- the relationship between motivational and cognitive components.

The survey results in motivation, in addition to severely violating normality, also suffered from a ceiling effect. A ceiling effect occurs when "a measure possesses a distinct upper limit for potential responses and a large concentration of participants score at or near this limit" (Hessling, Traxel, & Schmidt, 2004). One hypothesis regarding the ceiling effect is that the students, due to self-selection, already had high interest and value of the domains in question. As opposed to a random assignment, in self-selected quasi-experimental settings, the participants can possess

characteristics that can systematically confound the results, or in this case produce a ceiling effect.

The results of the motivation surveys can also be interpreted from a positive perspective. It is highly likely that, because the students self-selected to be a part of the GDL intervention, they had initially high levels of interest and utility value for game design, programming, and problem solving. In the post-survey, the students' interest and value levels either stayed the same or went up to a small degree. This might indicate that GDL intervention was effective in keeping students' motivations high. According to Hidi and Renninger (2006), students with high levels of individual interest opt to re-engage with tasks they are interested in, they try to develop a fuller understanding of the task by asking "curiosity" questions, and they try to exceed task demands. Although a systematic account of the participants cannot be provided in this study and it was beyond its scope, anecdotally it is correct to argue that these characteristics were applicable to the GDL participants in how they were interested in game design.

Limitations

First and foremost, this study was limited by its design as a quasi-experimental study. The subjects attended at their own will and that might have caused some systematic selection issues. Although the control group was also selected from a similar set of individuals (students who opted to join a summer version of the GDL course), this aspect of the research should be taken into consideration while generalizing the results to different populations.

Due to the small sample size and significant levels of non-normality in the data, I could not answer the research questions regarding motivational changes in this research. They are, however, important research questions to be tackled in future research.

In addition, the length of the motivation survey was considerably long for this age group and they were taken from research with undergraduate students. For these reasons, the survey items might have been difficult for the sample in this study to understand and the test-fatigue might have impacted students' responses. Similarly, because the problem-solving test was designed for 15-year-olds, the test might have been difficult for some of the younger students.

Because the sample for this study came from a young group of students, the repeated implementation of surveys asking for their self-perceptions was problematic. Especially, in a quasi-experimental study like this (i.e., self selected to be in the GDL courses), the students are very likely to rate themselves very high in the motivational constructs at time 1, causing ceiling effects and potential regression to the mean problems. Therefore, methods to overcome this should be developed, possibly incorporating newer technologies such as personal tablet computers.

Finally, another limitation of this study was that, due to the quantitative nature of the research, individual differences and subtleties in learning might have been overlooked. Therefore, studies that will look at these qualitative aspects of learning at GDL courses are also needed.

Conclusions and Implications

Teaching students important cognitive skills using computers have received a lot attention from researchers for the last three decades (Clements, 1999). Starting with the seminal work of Seymour Papert (1980), educational researchers have looked at ways to use the learning of computer programming as a gateway to more generalizable cognitive skills, especially for children (Mayer & Wittrock, 1996). A plethora of programming software simplified for young children (i.e., visual programming interfaces with drag and drop programming) and after-school

workshops have been developed and offered, with the promise of increased interest in STEM (science, technology, engineering and math) and computer science, and increased cognitive skills to help students to be successful in these domains. Early research efforts in understanding the connection between programming (i.e., LOGO) and higher order thinking skills, however, was often fraught with methodological limitations, often pointing to mixed findings (Mayer & Wittrock, 1996; D. B. Palumbo, 1990).

Recently, game design has been championed as the replacement of programming, and lauded as a way to scaffold computational thinking (Weintrop & Wilensky, 2012) and as a method of encouraging STEM careers. This popularity has also received official recognition by the U.S. Government when young game designers were invited to present their games created in Kodu at the Whitehouse Science Fair (Microsoft Citizenship Team, 2013). Such initiatives point to the increasing need for research to understand the actual impacts of learning game design and significance of this study. Research in this domain, however, has been either purely qualitative and anecdotal at best (e.g., Games & Kane, 2011; Richards & Wu, 2011; Wu & Richards, 2011) or lacking entirely. Therefore, this study is the first in recent literature that establishes an empirical link between game design and problem solving.

The results of this study have also important implications for theory and practice. First and foremost, the results indicated that, through a curriculum and instruction using effective methods of teaching problem solving, game design could foster complex problem-solving skills. Specifically, it was seen in this research that game design tasks could be aligned with multiple methods of teaching problem solving, such as structure-based methods, teaching by analogy, and teaching skills directly. It was also seen that through game design process, teaching of multiple types of problems is also possible. Most importantly, teaching system analysis and design,

decision-making and troubleshooting align very well with the underlying skills required for game design. Therefore, based on the results of this research, by carefully establishing *design* and *problem solving* as peer domains (Nelson, 2003), using game design is supported as a viable approach for teaching problem solving.

Another implication of this study is that this engaging method of teaching problem solving is suitable for different age groups, both males and females, and students' with differing initial game design or problem-solving experience. Within the scope of GDL initiative, game design courses were offered to students who were between 11 and 16, and all age groups equally benefitted from the GDL courses. In addition, girls as well as boys also attended the courses and, again, both groups enjoyed attending the courses and benefitted from the activities offered. Finally, GDL courses were also offered to students from different socio-economic backgrounds and different countries, and these groups also benefitted from being a part of the initiative. This, however, does not mean that every individual benefitted from the courses to exactly the same degree. There can be differences in how much different groups or individual students benefit from the courses due to some inherent characteristics that they have. Looking at these specific differences as independent variables was beyond the scope of this research. In order to adjust courses for every student's specific needs, and therefore ensure everyone benefits from the courses at comparable amounts, these factors should be investigated in future research.

Another practical implication of this study is that game design courses stand out as ideal alternatives to replace computer literacy courses at schools. By expanding the GDL curriculum to incorporate different thinking skills, these new courses introduce students to computers and computer programming, as well as also learning thinking skills that can be used outside of the game design context. Such initiative would also help trigger interest in students for STEM

degrees and careers, due to the tight connection between these technology-rich activities and what most STEM degrees and careers involve.

The effectiveness of GDL curriculum in teaching problem-solving skills also suggests that teaching of content knowledge can also be possible by putting a content layer into game design and problem-solving tasks. For example, important real-life issues such as environmental problems can be seamlessly integrated into the existing GDL curriculum, and game design scenarios. This could possibly help increase environmental awareness in students and lead to increased knowledge and interest in these issues in the future. Such an initiative could also trigger interest for science degrees and careers.

In terms of implications specifically for similar future interventions, it can be put forward that in GDL curriculum the aim was twofold: a) to first help students learn how to use the software, and b) then move to embedding the targeted skills into game design activities. This idea of skill building and the way of progression in the GDL curriculum can serve as a model for future interventions seeking similar goals. Early research in programming also supported the idea of starting with basics and guiding students into showing the connection between programming and thinking skills (Dalbey & Linn, 1985; Linn, 1985; Mayer, 1979). This idea is also based on "constraint-removal" methods of teaching problem solving, where mastery of basic skills enables problem solvers to focus more on the problem and be more successful due to automatization (Mayer & Wittrock, 1996). In the case of this study, after mastering the software, learners felt more comfortable during the design process, and this helped them to more successfully create problem scenarios in the game design environment. In addition, such a method helps learners build confidence over time and prevents them from feeling overwhelmed.

Learning though design is a very powerful method of learning (Harel, 1991). Through design, children get invaluable chances to play and tinker with objects and finally create something they value. Through this process they learn about their own learning and thinking (Papert, 1980). This research supported this ethos and established that game design can be a good platform for the teaching of problem-solving skills, but only when the methods of teaching are carefully designed to incorporate both theories of problem solving and effective methods of teaching game design.

Future Research

First, since GDL curriculum incorporated various methods of teaching problem solving, future studies are needed where each intervention benefits from a specific method. This would allow identifying the specific impact of different methods.

Statistical analyses pointed to differences in the gains within the experimental sites in the overall gains in problem solving and also gains in specific problem solving skills. As expected, the groups who started high also ended up higher than the others. The groups starting low, although having made significant gains compared to their pre-test scores, still finished at lower levels than the others. Therefore, future research should look into understanding potential independent variables such as socio-economic status, gender, and nationality to understand whether there are differences due to these variables. In addition, the specific connection between GDL tasks and specific problem-solving skills should also be studied.

At an individual level, there were students who did not make a lot of progress, and similar to Mayer & Fay (1987), it can be argued that there is a chain of cognitive changes promoted by learning to design games. It may be that certain student characteristics (e.g., self-regulation) predispose them to make significant gains. These characteristics were not within the

scope of the current study, and should be investigated in the coming studies. Some potential candidates for these characteristics (as independent variables) can be the students' self-regulation perceptions, and their goal orientations. Jonassen (2000) also argues that individuals differ in their problem-solving abilities due to possible differences in familiarity, domain and structural knowledge, cognitive controls, metacognition, epistemological beliefs, and, affective and conative (i.e., motivational and volitional). These domains can also be sources of individual differences and should be investigated further as factor impacting how much the students are benefiting from the GDL courses.

In order to better understand the specifics the learning process at GDL courses, other types of data should be collected during the studies. For example, observations specifically looking at certain student behaviors (i.e., asking for help, collaborative attitudes) can help identify the intricacies of the learning progression at GDL courses.

The motivational data for this research did not produce significant outcomes and it suffered from violating assumptions of normality and ceiling effects. One reason for the ceiling effect can be that students did not know that there would be a post-survey, and overrated themselves in the pre-survey. Methods should be developed to overcome such overestimations. Also, since this was a quasi-experimental study, students might have indeed come from such populations of highly motivated individuals. Therefore, true experiments can have the power to produce significant results.

Another future research direction that naturally follows from this study is to analyze student games to look for connections between gains in problem solving and level of complexity in games. This line of work can potentially help researchers understand the specific connections between game design and problem solving. Similarly, this line of work can lead to development

of interactive problem solving assessments that are built into the game design software and implemented during the GDL courses seamlessly.

Finally, just as well as embedding of thinking skills, content knowledge can also be embedded in GDL curriculum. For example, the scenarios given to students can also address important environmental problems, and can teach students environmental literacy and raise awareness. Future research can investigate this connection and evaluate the potential of such tasks in teaching content skills, raise awareness, and teach thinking skills simultaneously. Such research can possibly also look at the changes in students' interest and utility value of careers in science (or STEM) fields.

APPENDICES

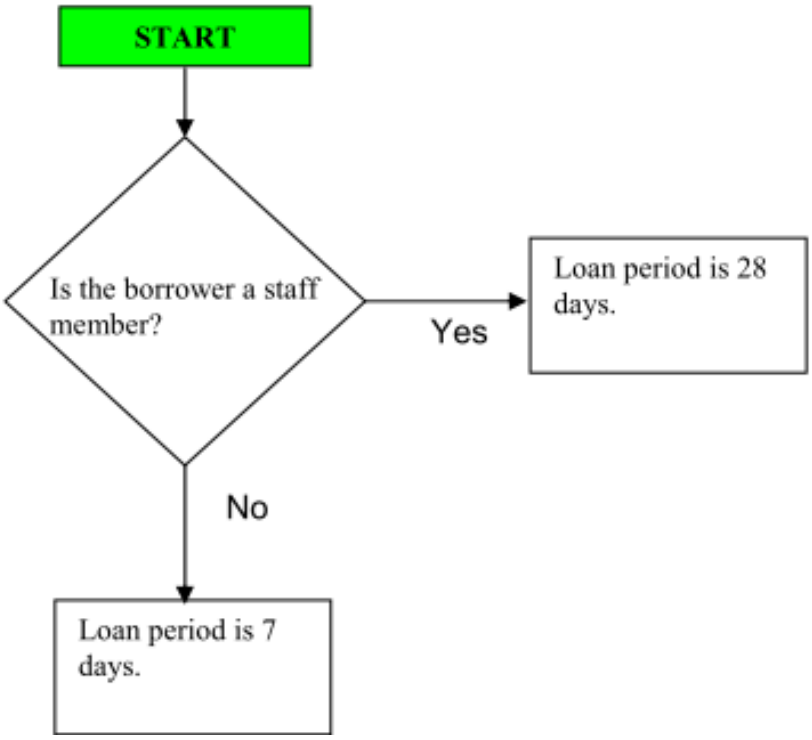
Appendix A

PISA 2003 Problem solving Test Sample Items

LIBRARY SYSTEM

The **John Hobson High School** library has a simple system for lending books: for staff members the loan period is 28 days, and for students the loan period is 7 days The following is a decision tree diagram showing this simple system:

Figure 21. *Library System of Hobson High School (PISA Sample Item)*



The **Greenwood High School** library has a similar, but more complicated, lending system:

- All publications classified as "Reserved" have a loan period of 2 days.
- For books (not including magazines) that are **not** on the reserved list, the loan period is 28 days for staff, and 14 days for students.
- For magazines that are **not** on the reserved list, the loan period is 7 days for everyone.
- Persons with any overdue items are not allowed to borrow anything.

Question 1: LIBRARY SYSTEM

You are a student at **Greenwood High School**, and you do not have any overdue items from the library. You want to borrow a book that is **not** on the reserved list. How long can you borrow the book for?

Answer: ..... days.

Question 2: LIBRARY SYSTEM

Develop a decision tree diagram for the **Greenwood High School Library** system so that an automated checking system can be designed to deal with book and magazine loans at the library. Your checking system should be as efficient as possible (i.e. it should have the least number of checking steps). Note that each checking step should have only **two** outcomes and the outcomes should be labelled appropriately (e.g. “Yes” and “No”).

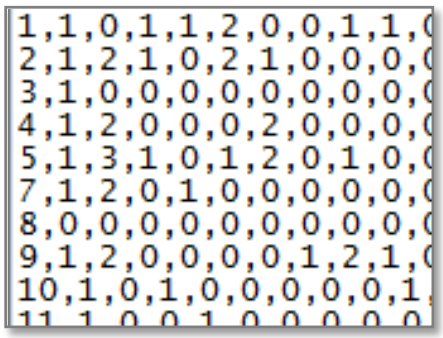
Appendix B

Analysis of PISA Scores Using IRT Methods

There are two steps of analyzing the PISA assessment to calculate the students' pre and post problem-solving scores: a) grading student tests based on instructions by PISA, and b) running item-response-theory (IRT) analyses to calculate each student's pre and post problem-solving proficiency.

First, grading was done using the answer key provided by PISA (OECD, 2004b). In accordance with the information provided by PISA, the dichotomous questions, or the questions with only one possible answer (i.e. multiple-choice questions), were coded as 0 when the given answer is incorrect, and as 1 when it is correct. For the polytomous items, or the items with a possible partial credit, scores were assigned as 0, 1, or 2, depending on if the question is wrong (0), partially correct (1) or fully correct (2). Going through this process, a data-matrix file containing each student's answers in numerical format was created (Figure 23).

Figure 22**.** *Screenshots of a Sample Data-Matrix File (Top) and Item Parameter File (Bottom)*



```
1,1,0,1,1,2,0,0,1,1,(
2,1,2,1,0,2,1,0,0,0,(
3,1,0,0,0,0,0,0,0,0,(
4,1,2,0,0,0,2,0,0,0,(
5,1,3,1,0,1,2,0,1,0,(
7,1,2,0,1,0,0,0,0,0,(
8,0,0,0,0,0,0,0,0,0,(
9,1,2,0,0,0,0,1,2,1,(
10,1,0,1,0,0,0,0,0,1,
11,1,0,0,1,0,0,0,0,0,
```

preX402Q01	1	2	1	A	P	-1.33		
preX402Q02	+	4	1	A	P	1.48	1.09	0.15
preX412Q01	1	2	1	A	P	-0.07		
preX412Q02	1	2	1	A	P	0.03		
preX412Q03	+	3	1	A	P	0.42	1.04	-1.04
preX414Q01	+	3	1	A	P	0.77	1.14	-1.14
preX415Q01	+	3	2	A	P	1.37	-0.40	0.40
preX417Q01	+	3	1	A	P	0.47	-0.44	0.44
preX423Q01	1	2	3	A	P	0.02		
preX423Q02	1	2	3	A	P	0.28		

In the second step of calculating problem-solving scores for each student, the item-related information from PISA 2003 Technical Report (OECD, 2005a)was used. By utilizing IRT procedures (*Rasch Partial Credit Model*) the students' problem-solving proficiency at the beginning and end of the camp is calculated.

IRT provides remedies for three important shortcomings of classical test theory (CTT) (Hambleton, 1990). Firstly, in CTT, item parameters are estimated based on the sample at hand, and are not valid for different samples. Secondly, in CTT, examinee ability estimates can be compared to another sample only when the full test or a very similar test is applied to the second sample, making it very difficult to create short tests that alternate items. Finally, in CTT, information regarding what an examinee might do when confronted with a specific item is not known, whereas in IRT "[a]bility estimates obtained from different item samples for an examinee will be the same except for measurement errors. This feature is obtained by incorporating information about the items (i.e., their statistics) into the ability estimation

process" (Hambleton, 1990, p. 99). Hence, by using IRT, we get individual proficiency estimates that are calculated based on well-established item difficulty parameters that were converged from previous data. These scores can be compared to data collected and analyzed using the same items. The Rasch model, the IRT model used by PISA and will be used for this study, uses only the item difficulty parameter.

Acknowledging that persons and items differ in difficulty and proficiency, IRT models "attempts to model the relationship between an unobserved variable, usually conceptualized as an examinee's ability, and the probability of the examinee correctly responding to any particular test item" (Harris, 1989, p. 157). Therefore, IRT models use a logistics function, turning dichotomous variables into continuous variables (Harris, 1989; OECD, 2005b), hence, making it possible to calculate a given student's probability of answering a question correctly. In addition, as suggested by OECD (2005b) in order to decrease the bias in items' abilities in reporting examinee proficiencies, a Weighted Likelihood Estimates (WLEs) will be applied.

After creating a student data matrix and an item-parameter file which has item-difficulty information (prepared using PISA parameters), the students' individual pretest and posttest problem-solving scores were calculated by an IRT software used and suggested by OECD (2005a), Xcalibre 4.1.6.1 (Assessment Systems Corporation, 2012).

Appendix C

GDL Motivation Survey

Please indicate how much you agree with the following statements by putting a number from 0 to 100 in the provided boxes. Remember that 100 indicates a very high degree of agreement, while 0 means certain disagreement. 50 would refer to "neutral".

Feel free to use scores 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, or scores like 78, 65, or 92.

For example, for an example statement of "I can play basketball well" we expect basketball players like Michael Jordan to give a 100.

Course-Enjoyment

	Your score of agreement (0 to 100):
I had fun at this course.	
Overall I am satisfied with this course.	
I am disappointed with this course.	
Instead of attending this course, I wish I had worked on my own.	
Course was a good use of my time.	

Computer and Technology Competence

	Your score of agreement (0 to 100):
Other people come to me for advice when they have questions about technology and computers.	
I am able to learn how to use new technologies rather quickly.	
I consider myself to be a highly skilled computer user.	
Compared to my peers, I am extremely skillful at using technology.	
Thinking about learning a new technology (e.g., a new software, photo editing, digital camera, etc.), it is likely I could teach myself how to efficiently use it.	
Compared to most other topics, technology is easy for me to learn.	

Game Design

	Your score of agreement (0 to 100):
I believe learning game-design is of some value to me.	
I think that learning game-design is useful for my future career.	
I think learning game-design is important because it can help me find a job in the future.	
I am willing to come to a similar game-design course again because it has some value to me.	
I think learning game-design helps me to be successful in my life.	
I believe learning game-design is beneficial to me.	
I think learning game-design is an important activity.	
I think game-design is an interesting subject.	
I am not interested in game-design.	
I like learning about game-design in this course.	
I think learning game-design is interesting.	
I've always wanted to learn more about game-design.	
I'm certain I mastered the game-design skills taught at this camp.	
I'm certain I figured out how to do the most difficult game-design tasks at this camp.	
I was able to do almost all the game-design tasks in class when I didn't give up.	
Even if game-design was hard, I was able to learn it.	
I was able to create even the most complex game at this camp when I tried.	
My experience at this camp makes me want to learn more about game-design.	
I want to have a job that involves game-design some day.	
I plan on learning more about game-design even when I don't have to.	
I am not really interested in using game-design skills in my future career.	

Programming

	Your score of agreement (0 to 100):
I believe learning programming is of some value to me.	
I think that learning programming is useful for my future career.	
I think learning programming is important because it can help me find a job in the future.	
I am willing to come to a similar programming course again because it has some value to me.	
I think learning programming helps me to be successful in my life.	

I believe learning programming is beneficial to me.	
I think learning programming is an important activity.	
I think programming is an interesting subject.	
I am not interested in programming.	
I think I liked learning about programming in this course.	
I think learning programming is interesting.	
I've always wanted to learn more about programming.	
I'm certain I mastered the programming skills taught at this camp.	
I'm certain I figured out how to do the most difficult programming tasks at this camp.	
I was able to do almost all the programming tasks in class when I didn't give up.	
Even if programming was hard, I was able to learn it.	
I was able to code even the most complex program at this camp when I tried.	
My experience at this camp makes me want to learn more about programming.	
I want to have a job that involves programming some day.	
I plan on learning more about programming even when I don't have to.	
I am not really interested in using programming skills in my future career.	

Problem solving

	Your score of agreement (0 to 100):
I believe learning problem-solving is of some value to me.	
I think that learning problem-solving is useful for my future career.	
I think learning problem-solving is important because it can help me find a job in the future.	
I am willing to come to a similar problem-solving course again because it has some value to me.	
I think learning problem-solving helps me to be successful in my life.	
I believe learning problem-solving was beneficial to me.	
I think learning problem-solving is an important activity.	
I think problem-solving is an interesting subject.	
I am not interested in problem-solving.	
I think I liked learning about problem-solving in this course.	
I think learning problem-solving is interesting.	
I've always wanted to learn more about problem-solving.	
I'm certain I mastered the problem-solving skills taught at this camp.	
I'm certain I figured out how to do the most difficult problem-solving tasks at this camp.	
I was able to do almost all the problem-solving tasks in class when I didn't give up.	

Even if problem-solving was hard, I was able to learn it.	
I was able to solve even the most complex problems at this camp when I tried.	
My experience at this camp makes me want to learn more about problem-solving.	
I want to have a job that involves problem-solving some day.	
I plan on learning more about problem-solving even when I don't have to.	
I am not really interested in using problem-solving skills in my future career.	

REFERENCES

REFERENCES

Ackermann, E. (2001). Piaget's constructivism , Papert's constructionism : What's the difference? *Future of Learning Group Publication*, *5*(3), 1–11. doi:10.1.1.132.4253

Alice. (2012). Retrieved from http://www.alice.org/index.php

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

Bandura, A. (1982). Self-efficacy mechanism in human agency. *American Psychologist*, *37*(2), 122–147.

Bennett, D., & Monahan, P. (2013). NYSCI Design Lab: No bored kids! In M. Honey & D. E. Kanter (Eds.), *Design, make, play: Growing the next generation of STEM innovators* (pp. 34–49). New York: Routledge.

Brophy, J. (1999). Toward a model of the value aspects of motivation in education: Developing appreciation for particular learning domains and activities. *Educational Psychologist*, *34*(2), 75–85.

Brophy, J. (2008). Developing students' appreciation for what is taught in school. *Educational Psychologist*, *43*(3), 132–141. doi:10.1080/00461520701756511

Brophy, J. (2010). *Motivating students to learn* (3rd ed.). New York, NY: Routledge.

Bruner, J. (1961). The act of discovery. *Harvard Educational Review*, *31*, 21–32.

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

Choi, W., & Repman, J. (1993). Effects of Pascal and FORTRAN programming on the problem-solving abilities of college students. *Journal of Research on Computing in Education*, *25*(3), 290–302.

Clements, D. H. (1999). The future of educational computing research : The case of computer programming. *Information Technology in Childhood Education Annual*, (1), 147–179.

Dalbey, J., & Linn, M. (1985). The demands and requirements of computer programming: A literature review. *Journal of Educational Computing Research*, *1*(3), 253–274.

Dewey, J. (1913). *Interest and effort in education*. Boston, MA: Houghton Mifflin Company.

Dweck, C. S. (1999). Caution: Praise can be dangerous. *American Educator*, *1*(23), 1–5.

Fullerton, T. (2008). *Game design workshop*. 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.

Gallini, J. K. (1987). A comparison of the effects of LOGO and a CAI learning environment on skills acquisition. *Journal of Educational Computing Research*, *3*(4), 461–477.

Games, I. A. (2009). *21st century language and literacy in Gamestar Mechanic: Middle school students' appropriation through play of the discourse of computer game designers*. (Order No. AAI3384100, *Dissertation Abstracts International Section A: Humanities and Social Sciences*, 3745).

Games, I. A., & Kane, L. P. (2011). Exploring adolescent's STEM learning through scaffolded game design. *Proceedings of the 6th International Conference on Foundations of Digital Games* (pp. 1–8). New York: ACM. doi:10.1145/2159365.2159366

Gamestar Mechanic. (2012). Retrieved from http://gamestarmechanic.com

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology*, *12*, 306–355.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, (15), 1–38.

Glynn, S. M., Britton, H. K., Muth, D., & Dogan, N. (1982). Writing and revising persuasive documents: Cognitive demands. *Journal of Educational Psychology*, *74*(4), 557–567.

Guzdial, M. (2004). Programming environments for novices. In S. Fincher & M. Petre (Eds.), *Computer science education research* (pp. 1–16). The Netherlands, Lisse: Taylor & Francis.

Hambleton, R. K. (1990). Item response theory: Introduction and bibliography. *Psicothema*, *2*(1), 97–107.

Hansen, D. A. (1989). Lesson evading and lesson dissembling: Ego strategies in the classroom. *American Journal of Education*, *97*, 184–208.

Harel, I. (1991). *Children designers: Interdisciplinary constructions for learning and knowing mathematics in a computer-rich school*. Ablex Publishing Corporation.

Harris, D. (1989). Comparison of 1-, 2-, and 3-Parameter IRT Models. *Educational Measurement: Issues and Practice*, *8*(1), 35–41.

Hessling, R. M., Traxel, N. M., & Schmidt, T. J. (2004). Ceiling effect. In M. S. Lewis-Beck, A. Bryman, & T. F. Liao (Eds.), *The SAGE encyclopedia of social science research methods*. Sage Publications Inc. doi:10.4135/9781412950589

Hidi, S., & Renninger, K. (2006). The four-phase model of interest development. *Educational Psychologist*, *41*(2), 111–127. Retrieved from

Huber, O. (1995). Complex problem solving as multi stage decision making. In P. A. Frensch & J. Funke (Eds.), *Complex problem solving: The European perspective* (pp. 151–173). Hillsdale, NJ: Lawrence Erlbaum Associates.

Hulleman, C. S., Durik, A. M., Schweigert, S. B., & Harackiewicz, J. M. (2008). Task values, achievement goals, and interest: An integrative analysis. *Journal of Educational Psychology*, *100*(2), 398–416. doi:10.1037/0022-0663.100.2.398

Hulleman, C. S., Godes, O., Hendricks, B. L., & Harackiewicz, J. M. (2010). Enhancing interest and performance with a utility value intervention. *Journal of Educational Psychology*, *102*(4), 880–895. doi:10.1037/a0019506

Hulleman, C. S., & Harackiewicz, J. M. (2009). Promoting interest and performance in high school science classes. *Science*, *326*(5958), 1410–2. doi:10.1126/science.1177067

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development*, *48*(4), 63–85.

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

Kafai, Y. B., Carter Ching, C., & Marshall, S. (1997). Children as designers of educational multimedia software. *Computers & Education*, *29*(2-3), 117–126. doi:10.1016/S0360-1315(97)00036-5

Kafai, Y. B., Peppler, K. A., & Chapman, R. N. (2009). *The computer clubhouse: Constructionism and creativity in youth communities*. New York: Teachers College Press.

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.

Lehrer, R., Lee, M., & Jeong, A. (1999). Reflective teaching of LOGO. *The Journal of the Learning Sciences*, *8*(2), 245–289.

Lepper, M. (1985). Microcomputers in education: Motivational and social issues. *American Psychologist*, *40*(1), 1–18.

Liao, Y. C. (2000). A meta-analysis of computer programming on cognitive outcomes: An updated synthesis. In J. Bourdeau & R. Heller (Eds.), *Proceedings of world conference on educational multimedia, hypermedia and telecommunications* (pp. 598–604). Chesapeake, VA: AACE.

Liao, Y. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research*, *7*(3), 251–266.

Linden, M., & Wittrock, M. C. (1981). The teaching of reading comprehension according to the model of generative learning. *Reading Research Quarterly*, *17*(1), 44–57.

Linn, M. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher*, *14*(5), 14–29.

Littlefield, J., Delclos, V. R., Bransford, J. D., Clayton, K. N., & Franks, J. (1989). Some prerequisites for teaching thinking : Methodological issues in the study of LOGO programming. *Cognition and Instruction*, *6*(4), 331–366.

MacLaurin, M. B. (2011). The design of Kodu: A tiny visual programming language for children on the Xbox 360. *ACM SIGPLAN Notices*, *46*(1).

Maehr, M. L., & Meyer, H. A. (1997). Understanding motivation and schooling: Where we've been, where we are, and where we need to go. *Educational Psychology Review*, *9*(4), 371–409.

Maloney, J., Burd, L., Kafai, Y. B., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: A sneak preview. *International conference on creating, connecting and collaborating through computing* (pp. 104–109). Ieee. doi:10.1109/C5.2004.1314376

Mayer, R. E. (1977). *Thinking and problem solving: An introduction to human cognition and learning*. Scott, Foresman and Company.

Mayer, R. E. (1979). A psychology of learning BASIC. *Communications of the ACM*, *22*(11), 589–593. doi:10.1145/359168.359171

Mayer, R. E. (1998). Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science*, *26*(1), 49–63.

Mayer, R. E. (1999). Multimedia aids to problem-solving transfer. *International Journal of Educational Research*, *31*.

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), 14–9. doi:10.1037/0003-066X.59.1.14

Mayer, R. E., & Fay, A. L. (1987). A chain of cognitive changes with learning to program in Logo. *Journal of Educational Psychology, 79*(3), 269.

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, NY: 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). Mahwah, NJ: Lawrence Erlbaum Associates.

Microsoft Citizenship Team. (2013). Young Kodu designer showcases at 2013 White House Science Fair. *blog.technet.com*. Retrieved May 4, 2013, from http://blogs.technet.com/b/microsoftupblog/archive/2013/04/24/2013-white-house-science-fair.aspx

Microsoft Kodu. (2012). Microsoft Research. Retrieved from http://research.microsoft.com/en-us/projects/kodu/

Mitchell, M. (1993). Situational interest: Its multifaceted structure in the secondary school mathematics classroom. *Journal of Educational Psychology, 85*(3), 424–436. doi:10.1037//0022-0663.85.3.424

Moreno, R. (2004). Decreasing cognitive load for novice students: Effects of explanatory versus corrective feedback in discovery-based multimedia. *Instructional Science, 32*, 99–113.

Moreno, R., & Mayer, R. E. (1999). Multimedia-supported metaphors for meaning making in mathematics. *Cognition and Instruction, 17*(3), 215–248.

Nastasi, B. K., Clements, D. H., & Battista, M. T. (1990). Social-cognitive interactions, motivation, and cognitive growth in Logo programming and CAI problem-solving environments. *Journal of Educational Psychology, 82*(1), 150.

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. (2003). *PISA 2003 sssessment framework: Mathematics, reading, science and problem solving knowledge and skills*. Paris.

OECD. (2004a). *Problem solving for tomorrow's world: First measures of cross-curricular competencies from PISA 2003*. Paris: OECD Publishing.

OECD. (2004b). *PISA 2003 problem solving answer key* (Vol. 659). Paris: OECD Publishing. Retrieved from http://www.oecd.org/document/38/0,3746,en_32252351_32236173_34993126_1_1_1_1,00.html

OECD. (2005a). *PISA 2003 technical report*. Paris: OECD Publishing.

OECD. (2005b). *PISA 2003 data analysis manual: SPSS users*. Paris. Retrieved from http://www.oecd.org/dataoecd/35/51/35004299.pdf

OECD. (2012). *PISA 2012 field trial problem solving framework*. Paris. Retrieved from http://www.oecd.org/dataoecd/8/42/46962005.pdf

Pajares, F. (1996). Self efficacy beliefs in academic settings. *Review of Eduational Research*, *66*(4), 543–578.

Palumbo, D. B. (1990). Programming language / Problem-solving a review of relevant issues. *Review of Educational Research*, *60*(1), 65–89.

Palumbo, D. L., & Palumbo, D. B. (1993). A comparison of the effects of Lego TC Logo and problem solving software on elementary students' problem solving skills. *Journal of Computing in Childhood Education*.

Palumbo, David B. (1990). Programming language / Problem-solving a review of relevant issues. *Review of Educational Research*, *60*(1), 65–89.

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

Papert, S. (1994). *The children's machine: Rethinking school in the age of the computer*. Basic Books.

Papert, S., & Harel, I. (1991). Situating constructionism. In S Papert & I. Harel (Eds.), *Constructionism* (Vol. 36, pp. 1–11). Norwood, NJ: Ablex Publishing Corporation. doi:10.1111/1467-9752.00269

Pausch, R., Burnette, T., Capeheart, A. C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., et al. (1995). Alice: Rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications*, *15*(3), 8–11.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *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.

Phillips, M. M. (2007). *The influence of situational factors on the nurturance of personal interest and perceived competence*. (Order No. AAI3298099, *Dissertation Abstracts International Section A: Humanities and Social Sciences*).

Polya, G. (1957). *How to solve it*. Garden City, NY: Doubleday/Anchor.

Quilici, J. L., & Mayer, R. E. (2002). Teaching students to recognize structural similarities between statistics word problems. *Applied Cognitive Psychology*, *16*(3), 325–342.

Reed, W. M., & Palumbo, D. B. (1992). The effect of BASIC instruction on problem solving skills over an extended period of time. *Journal of Educational Computing Research*, *8*(3), 311–325.

Repenning, A. (1993). *Agentsheets: A tool for building domain-oriented dynamic, visual environments.* (Order No. 9423532, University of Colorado at Boulder). *ProQuest Dissertations and Theses.*

Repenning, A. (2012). AgentSheets. AgentSheets, Inc. Retrieved from http://www.agentsheets.com/

Resnick, L. B. (1987). *Education and learning to think*. National Academies Press.

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

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., et al. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

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.

Resnick, M., & Rusk, N. (1996). The Computer Clubhouse: Preparing for life in a digital world. *IBM Systems Journal*, *35*(3.4), 431–439.

Richards, K., & Wu, M. L. (2011). Examining digital game-based learning through the lens of 21st century gamers. In M. J. Koehler & P. Mishra (Eds.), *Proceedings of society for information technology & teacher education international conference* (pp. 45–52). Chesapeake, VA: AACE.

Salomon, G., & Perkins, D. N. (1987). Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research*, *3*(2), 149–169.

Scratch. (2012). Retrieved from http://scratch.mit.edu/

Shin, T. S. (2010). *Effects of providing a rationale for learning a lesson on students' motivation and learning in online learning environments*. Order No. 889930919, Michigan State University). *ProQuest Dissertations and Theses*

Stolee, K. T., & Fristoe, T. (2011). Expressing computer science concepts through Kodu Game Lab. *Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 99–104).

Suomala, J., & Alajaaski, J. (2002). Pupils' problem-solving processes in a complex computerized learning environment. *Journal of Educational Computing Research*, *26*(2), 155–176.

Torres, R. J. (2009). *Learning on a 21st century platform: Gamestar mechanic as a means to game design and systems-thinking skills within a nodal ecology.* (Order No. AAI3361988, *Dissertation Abstracts International Section A: Humanities and Social Sciences*)

Utting, I. A. N., Maloney, J., & Resnick, M. (2010). Alice, Greenfoot, and Scratch – A Discussion. *ACM Transactions on Computing Education*, *10*(4), 1–11. doi:10.1145/1868358.1868364

Weiner, B. (1985). An attributional theory of achievement motivation and emotion. *Psychological Review*, *92*(4), 548–573.

Weintrop, D., & Wilensky, U. (2012). RoboBuilder: Video game program-to-play constructionist. *Constructionism 2012* (pp. 1–5). Athens, Greece.

Wigfield, A. (1994). Expectancy-value theory of achievement motivation: A developmental perspective. *Educational Psychology Review*, *6*(1), 49–78.

Wigfield, A., & Eccles, J. S. (2000). Expectancy-value theory of achievement motivation. *Contemporary Educational Psychology*, *25*(1), 68–81. doi:10.1006/ceps.1999.1015

Wigfield, A., & Eccles, J. S. (2004). Expectancy value theory in cross-cultural perspective. In D. M. McInerney & S. Van Etten (Eds.), *Big theories revisited* (pp. 165–198). Information Age Publishing.

Wigfield, A., Eccles, J. S., Roeser, R. W., & Schiefele, U. (2008). Development of achievement motivation. In W. Damon & R. M. Lerner (Eds.), *Child and adolescent development: An advanced course* (pp. 406–434). New Jersey: John Wiley and Sons.

Woodward, J., Carnine, D., & Gersten, R. (1988). Teaching problem solving through computer simulations. *American Educational Research Journal*, *25*(1), 72.

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.

Ziegler, E. W., & Terry, M. S. (1992). Instructional methodology, computer literacy, and problem solving among hifted and talented students. *International Journal of Instructional Media*, *19*(1), 45–51.