3-2017

Pencil Puzzles for Introductory Computer Science: an Experience- and Gender-Neutral Context

Zachary Butler
Rochester Institute of Technology

Ivona Bezáková
Rochester Institute of Technology

Kimberly Fluet
St. John Fisher College

Recommended Citation
Zack Butler, Ivona Bezakova, and Kimberly Fluet. 2017. Pencil Puzzles for Introductory Computer Science: an Experience- and Gender-Neutral Context. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). ACM, New York, NY, USA, 93-98. DOI: https://doi.org/10.1145/3017680.3017765

Pencil Puzzles for Introductory Computer Science: an Experience- and Gender-Neutral Context

Zack Butler and Ivona Bezáková
Computer Science
Rochester Institute of Technology
Rochester, NY 14623 USA
{zjb,ib}@cs.rit.edu

Kimberly Fluet
Math, Science and Technology Education
St. John Fisher College
Rochester, NY 14618 USA
kfluet@sjfc.edu

ABSTRACT

The teaching of introductory computer science can benefit from the use of real-world context to ground the abstract programming concepts. We present the domain of pencil puzzles as a context for a variety of introductory CS topics. Pencil puzzles are puzzles typically found in newspapers and magazines, intended to be solved by the reader through the means of deduction, using only a pencil. A well-known example of a pencil puzzle is Sudoku, which has been widely used as a typical backtracking assignment. However, there are dozens of other well-tried and liked pencil puzzles available that naturally induce computational thinking and can be used as context for many CS topics such as arrays, loops, recursion, GUIs, inheritance and graph traversal. Our contributions in this paper are two-fold. First, we present a few pencil puzzles and map them to introductory CS concepts that the puzzles can target in an assignment, and point the reader to other puzzle repositories which provide the potential to lead to an almost limitless set of introductory CS assignments. Second, we have formally evaluated the effectiveness of such assignments used at our institution over the past three years. Students reported that they have learned the material, believe they can tackle similar problems, and have improved their coding skills. The assignments also led to a significantly higher proportion of unsolicited statements of enjoyment, as well as metacognition, when compared to a traditional assignment for the same topic. Lastly, for all but one assignment, the student's gender or prior programming experience was independent of their grade, their perceptions of and reflection on the assignment.

Keywords

Learning in context; computational thinking; pencil puzzles; introductory CS concepts

1. INTRODUCTION

Puzzles have long been a natural inspiration for computer science assignments due to the computational thinking process they are designed to induce. However, despite ubiquitous use of the well-known Sudoku puzzle for backtracking assignments, the variety of such assignments is rather small, and, to the best of our knowledge, the efficacy of these assignments has not been formally studied. In this work we tap into the rich resource of pencil puzzles to design different assignments suitable for introductory computer science, targeting various CS concepts. We used these assignments over the past three years in the CS1/2 courses at our institution and evaluated them through lenses of gender, experience, student perceptions, and student reflective thinking.

Pencil puzzles are popular throughout the world, published in all cultures and solved by people of any age, gender, or race. They range from the well-known crosswords and Sudoku puzzles to lesser known but also much liked puzzles, some of which we describe in this paper. Generally speaking, pencil puzzles are those solved with a deductive, algorithmic process, typically on paper, with short and well-defined rules and unique solutions. While Sudoku and a couple of other puzzle types have been used for some CS topics, there are many different types of these puzzles available in the wild to provide fresh content, and they can be used as context for many more topics than may be immediately obvious.

Pencil puzzles have several features that make them especially well suited as the basis for CS assignments: (1) Due to space requirements by publishers, they need to be describable in a concise fashion. This means that the rules are succinct and simple to understand, leading to assignments with short and clear problem statements. (2) They naturally induce computational thinking—a puzzle and list of rules immediately suggests an algorithm and "wants" to be solved. (3) They have been tried, tested, and are liked—many such puzzles are published in multiple venues and with many authors and solvers. CS assignments sometimes suffer when the instructors, endeavoring to create a novel assignment targeting a specific topic, come up with a very artificial and confusing scenario. Using a well-tested puzzle avoids this pitfall. (4) There are a plethora of pencil puzzles to choose from. Having a large repository for assignment ideas allows instructors to create new, succinct, clear assignments term after term. (5) Most pencil puzzles are language and culture independent, and as such can appeal to a wide range of students. For example, Puzzle Picnic is a site based in the Netherlands with contributors and users from many countries. As of August 2016, contributors have posted over 5000 puzzles in 100 types, and over 7000 users have registered (registration is not required to play the puzzles). Of those users who have reported gender, 44% are female [12].

SIGCSE '17, March 08-11, 2017, Seattle, WA, USA

 ISBN 978-1-4503-4698-6/17/03…$15.00

DOI: http://dx.doi.org/10.1145/3017680.3017765

1.1 Related Work

Introductory Computer Science courses can be a challenge to deliver the content in a way that both promotes student learning and motivation. It is established that providing a real-world context for these abstract concepts can help students both stay interested and learn the underlying material [11, 14]. There are many definitions of "context-based learning"; in this work we mean providing a concrete example for teaching abstract computing concepts. In CS as well as across the sciences, the concept of context-based learning has been deployed using many different domains. For example, biology [4], computing hardware [16], media computation [18], board, card, and dice games [5, 8, 2], and even science fiction [1] have been applied to CS topics. Oliviera et al. compare context-based learning with abstract problem domains and report that "the type of domain (either concrete or abstract), when taken in and on itself, does not affect the learning of introductory programming," however, learning is influenced by whether students can relate to the domain or not, supporting the context-based learning paradigm [11, 14]. While there is little research on how different populations relate to pencil puzzles, in addition to the anecdotal reports from PuzzlePicnic, research has shown no effect of gender on the ability to solve Sudoku puzzles in populations of middle school students [10] and the elderly [3].

Based on our experience and interests, we have investigated the use of pencil puzzles as a context for introductory CS. These puzzles are somewhat different from other types of puzzle-based learning that have also been used to good effect in CS. The puzzle-based learning of Michalewicz et. al. [9, 19, 6] uses more open-ended puzzles to stimulate student thinking — we are concentrating on more well-defined puzzles with an explicit algorithmic aspect. Harder problems have been shown to promote computational thinking [17] but these problems can be too challenging for many new students. Another goal of using pencil puzzles is that we can take advantage of the large variety of puzzles to continually develop new assignments and use them for individual topics within a course as desired without having to invent new puzzles or redesign an entire course.

We have also been inspired by the concepts of problem-based learning, and the ability of non-computer-based activities to inspire computational thought. The goal of problem-based learning is to present students with a challenging, open-ended problem to encourage discussion and engagement with the material, and this has been implemented and researched extensively in many areas including K-12 settings [7]. CS unplugged[1] uses activities without electronic devices to teach algorithms and problem solving. Recent efforts have begun to systematically analyze their effectiveness for student learning of various CS concepts [15]. In our department's introductory courses, each assignment begins with a paper-based problem solving session but students then apply their results to create software implementation of the algorithms developed. This works well with a pencil puzzle, where the problem itself is naturally solved on paper before students develop algorithms and their implementations.

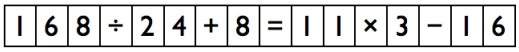
2. PENCIL PUZZLES AND ASSIGNMENTS

As discussed above, our goal is to develop and promulgate a context for CS learning that is reusable, intuitive and interesting to a wide variety of students, and we feel that pencil puzzles achieve these aims. In addition, they are well suited to individual stand-alone assignments, so that instructors may integrate them for any number of topics throughout a course. Here we provide several examples of different types of pencil puzzles, along with how we have used them (or suggest using them in the future) in the context of CS1 and/or CS2. In general, we develop our assignments for a given concept by determining a puzzle that would be suitable. The assignment itself typically starts by introducing the puzzle to the students, having them solve a simple instance, and then leading to a series of questions about the algorithms and data structures used in their implementation. Only a fraction of these assignments involve solving the puzzles, they may instead be asked to implement code that will verify a puzzle's solution or develop an interface for a human to solve it. Puzzles are also an excellent domain for discussions about testing one's code and how to design test cases. All of the puzzles and assignments presented here, along with many others, are publicly available in our repository [13]. In most cases here we refer to a particular resource for a given type of puzzle, but this may not be the only place to find them, and may also not be the original creator of the puzzle — in fact the creator is often unknown.

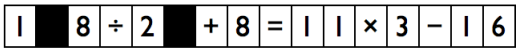
2.1 Examples

Blackout Math

In a Blackout Math puzzle[2], you are given a string of digits, math operations, and an equality sign. To solve the puzzle, black out (or delete) two of the characters to form a correct equality. For example, for



the solution is:



This puzzle can be used for assignments on a variety of topics and of different difficulty levels. For example, on the simpler end of the spectrum, as an assignment on one dimensional arrays, lists, or strings, and simple loops, we can restrict the input to only single-digit numbers (that is, we guarantee that the string has no consecutive digits) and just the addition and subtraction (+,−) operations. We can even provide the blacked out locations and ask the students to verify whether the provided solution is indeed correct. We can then add complexity by including multi-digit numbers and allowing other arithmetic operations (without priority of operators and parentheses). For nested loops, the students can find a solution (or all solutions!) on their own. Finally, as an assignment on stacks or trees, we can add parentheses and priority of operations. We have used this puzzle as the basis for an assignment multiple times in CS1, either as a single lab in one of its simpler forms, or, in its full complexity, as a term project. Very recently it was also used at a local high school.

KenKen

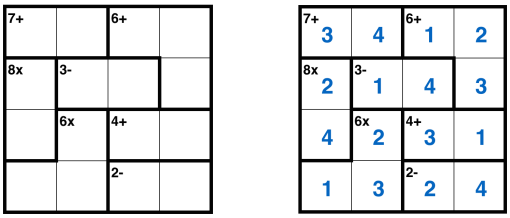
In a KenKen puzzle[3], you are given an $n \times n$ grid partitioned into regions. For each region you get a target value and for

[1] http://csunplugged.com

[2] http://www2.stetson.edu/~efriedma/published/blackout/index.html

[3] http://www.kenken.com

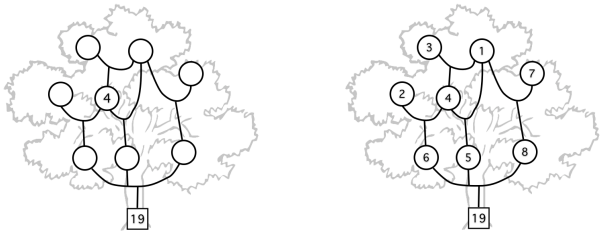
regions of two or more squares you also get an arithmetic operator. The goal is to fill the grid with the numbers 1 to n so that in each row and column each number appears exactly once, and in every region when you apply the corresponding arithmetic operation to the numbers in the region, you get the region's target value. A sample puzzle is shown on the left and its solution is on the right:



We based one of our CS2 inheritance assignments on this puzzle. Briefly, we asked the students to verify the correctness of a provided solution using different types of regions (additive, multiplicative, and so on) that inherit from a general region class. Additionally, this puzzle supports working with two dimensional arrays, and loops. In our assignments we listed the coordinates of all the squares in a region, to help the students concentrate on the inheritance aspect and not get stuck on parsing the input. Variants of the assignment include bounds on the sizes of the regions, types of allowed operations (for example, sum of even numbers in the region), or different represenations of input.

Number Tree

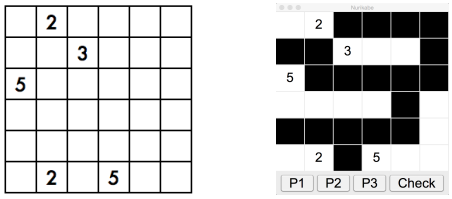
In a Number Tree puzzle, you are given a schematic figure resembling a tree with n circular locations, most or all of them empty, and a root node with a provided value, as shown on the figure on the left. You must assign the remaining numbers between 1 and n to the empty locations so that for each non-leaf its value is the result of adding up the values at the locations above which it is connected to. The solution to this example is on the right:



Depending on the structure of the figure, this can be an assignment on a tree or a directed acyclic graph traversal. If the number of leaves (or vertices of in-degree 0) is small, the students can search for a solution by examining all possibilities at just these vertices and propagating the values onward through the data structure, or even by generating all permutations of the numbers recursively and testing their assignments to the empty locations.

Nurikabe

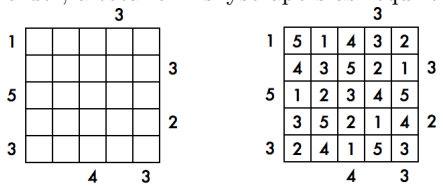
A Nurikabe puzzle[4] consists of a grid where some of the cells contain numbers. You must blacken some of the empty cells so that each white region contains exactly one cell with a number, which signifies the size of the white region. Meanwhile, the black cells must all be connected but without any 2×2 area of black. A sample puzzle is on the left, with its solution, shown as part of a GUI assignment, on the right.



This puzzle can lead to assignments on arrays and loops, or basic graph traversals to verify the size of each region and the connectedness of the black cells. We chose to use it as a basis for a GUI assignment, where the user tries to solve several hard-coded puzzles and the students provide a graphical interface that can switch between different puzzles and verify whether the user's solution is correct.

Skyscrapers

In a skyscraper puzzle[5], you are given a (usually) empty grid with some rows and columns marked with numbers on the outside of the grid. You fill in the grid cells with numbers that represent the height of a skyscraper at that location, such that no number repeats in a row or column. The numbers on the outside signify the number of different skyscrapers in that row or column that are visible from the location of the number. For example, for the input on the left, the solution is on the right: In the first row, looking from the left, we see only one skyscraper of height 5, it eclipses all other skyscrapers in that row. In the third column, looking from the bottom, we see skyscrapers of heights 1, 2, 3, and 5, in this order; a total of 4 skyscrapers as required.



This puzzle can be used as an exercise with loops and computational thinking — in verifying a single row or column, how do we determine the number of visible skyscrapers? Once the basic algorithm is discovered, depending on the level of the course, it can also lead to discussion of code reuse (using the same function to test each of the four directions) and data structure development (should the clues be in separate arrays, or as part of a single 2-D array?).

2.2 Mapping to introductory CS concepts

CS topic	Puzzle type(s)
Math expressions	24 puzzle
Conditionals	Skyscraper ; Mastermind
Lists and strings	Blackout Math
Nested loops	Easy as ABC; Skyscraper
Recursion	Jumble; Number Tree
Inheritance	KenKen
Data structures	Number Tree; Numerical Wheel
GUIs	Nurikabe
Graph Search	Hitori; Nurikabe; Mazes

[4]http://www.nikoli.com/en/puzzles/nurikabe/

[5]http://www.puzzlepicnic.com/genre?id=21

One of the major goals in our project is to use pencil puzzles to target as many standard introductory CS topics as possible. In the table above we show a number of puzzles that we have identified as relevant to particular topics — full descriptions of the puzzles not described above are available in our repository, along with ideas of how these puzzles might be used to teach the given concept.

2.3 Course Format and a Typical Assignment

In our department's introductory CS courses, we have been following a hybrid problem-solving model since 2009. Each week, we spend one hour of class time in a group-based problem solving session. During this session, student groups answer questions on paper related to the week's assignment, often including algorithm design and pseudocode. Some groups are asked to present their work to the class, so that all students can discuss their approaches and potentially modify their solutions. In the second hour of class time, students begin individual work on the programming portion of the assignment. We note that while this approach has been well-received by both our students and faculty, these puzzle-based assignments can be adapted to other forms of instruction, from purely homework-based assignments to studio approaches. Here we present a sketch of one of our assignments (using KenKen to explore inheritance in CS2); the full version is available in our repository.

Sample Assignment: KenKen verifier

In this lab you will write a complete implementation for a KenKen puzzle verifier. *Description and example of KenKen puzzles follow, much as in Sec. 2.* Given a puzzle with a potential solution, your program will verify the correctness of each region.

Problem Solving

1. What is the solution to this puzzle? *Puzzle shown above is given here.*
2. Let's assume that we have a division region whose target is 2 and whose values are stored in an ArrayList <Integer> in the order [2,4,1]. Write pseudocode for a general purpose boolean function, verify, that returns whether any supplied region is correct or not. You can assume the target and values are already accessible.
3. From a design perspective, what state and behavior is common to all regions?
4. Likewise, what state and behavior is unique to each region?
5. Design a UML diagram with an abstract Region class, and two subclasses, AddRegion (for performing addition), and DivideRegion (for performing division).

Implementation At this point in the lab writeup, students are given the details of the input file format and the expected output. They are then given the UML diagram for the classes that they are expected to implement, along with detailed Javadocs. Specifically, they are to implement the classes KenKen (main), Grid, Region (superclass), AddRegion, MultiplyRegion, SubtractRegion, and DivideRegion.

3. EVALUATION

At the start of this study, we conjectured that pencil puzzles are an effective problem domain for teaching introductory computer science across experience levels and genders.

Enjoy	I enjoyed this lab.
Frustrating	This lab was very frustrating.
BelieveICan	I believe I could write code that solves/ verifies similar puzzles.
Learn	I felt that this lab helped me to learn this week's material.
Coding skills	This lab/project helped improve my coding skills in Python/Java.

Table 1: Statements in surveys with which students were asked to agree or disagree. The left-hand column gives a short identifier used in the reporting of results for these statements.

positive	complimentary feedback
negative	critical feedback
balanced	combined complimentary and critical feedback
metacognitive	exhibits reflection on own learning

Table 2: Codes applied to free-form comments.

To evaluate these hypotheses, we implemented a number of pencil puzzle based assignments over three years and collected student survey data as well as grades on the assignments and in the course. Here we present our findings.

We have given two different assignments in two offerings of Computer Science 1 (CS1), in Python, and five different assignments in three offerings of Computer Science 2 (CS2), in Java. All of these assignments were developed by the authors of this paper or by course instructors in consultation with the authors. In the Fall of 2015, we were able to offer two different assignments to parallel sections of a course: the department offered two synchronized sections of CS2, and we gave a puzzle-based assignment in one section and a more traditional assignment for the same topic in the other.

For each offering, we surveyed the students immediately after the deadline of the assignment. Although these surveys evolved over time, all surveys included the same core questions shown in Table 1 to examine the students' perceptions of the assignment (Enjoy, Frustrating) and their reflections on their own learning related to the assignment (Learn, BelieveICan, Coding Skills). These questions were formed as statements with standard agree-disagree Likert scale responses and are given in full in Table 1.

Students were able to self-report gender (Male, Female, Both/Neither/prefer not to answer), experience level, and could provide open-ended comments about the lab or project. The open-ended comments were coded for emergent categories and converted to at most three of twelve codes by the authors. The analysis in this paper is based on the four codes shown in Table 2 because they were most relevant to investigating perception (Positive, Negative) and reflection by students (Balanced, Metacognitive).

In addition, at the close of the Spring 2016 semester we conducted focus groups and semi-structured interviews with the instructional teams (instructors, TAs, SLIs) for both CS1 and CS2. We are currently transcribing all the interviews and focus groups in preparation for a formal analysis and correlation with the survey data.

3.1 Instructional Environment

At RIT, CS1 and CS2 are required courses for students

majoring in Computer Science, Software Engineering, Computing Security, Computer Engineering, and Computational Mathematics, and are also taken by other majors. The enrollment is typically around 500—for example, in Spring 2016 we offered 10 sections of CS2 with a total of 457 students registered. Even in the "off" terms (that is, CS1 in the spring and CS2 in the fall), we offer several sections of the given course. Although we evaluated seven different assignments, due to space restrictions, here we present a selection of the data analysis over different courses in different years. In particular, we report results from two different offerings of CS1 in Fall 2014 and Spring 2016, and CS2 in Fall 2015. In the two offerings of CS1, we offered a lab and a project respectively, both based on Blackout Math. In Fall 2015, two sections of CS2 were taught, and each section was given a different lab assignment for inheritance—one section the KenKen assignment described above and the other a standard assignment based on different types of books in a bookstore. In total, 340, 80 and 64 students filled out the surveys in the three courses, with percentages of female students 12.6%, 15% and 11%. We note that while the population who reported gender as "both, neither, or prefer not to answer" was too small a sample to offer significance to gender analysis, these students were included in all remaining analyses. The full quantitative analysis of all surveys is available at our repository [13].

3.2 Data Analysis Procedures

Since our focus is on the accessibility of pencil puzzle projects and labs by a wide audience, we investigated if the labs and projects were *gender* and *experience neutral*, based on students' grades; their *perception* of the labs and puzzles (survey statements: Enjoy, Frustrating; comments: Positive, Negative) and the impact of the labs and puzzles on their *reflective thinking* (survey statements: Learn, CodingSkills, BelieveICan; comments: Balanced, Reflective). All analyses were performed using SPSS statistical software. A one-way Analysis of Variance (ANOVA) was conducted to determine the effect of gender and, separately, experience, on answers to the core survey statements with significance levels at $p < .05$, results of which are reported below. Descriptive statistics were calculated for survey statements and comments. Frequencies of responses are given in Table 3.

3.3 Survey Results and Analysis

Gender: Of particular interest was investigating whether the CS1/CS2 labs were gender neutral. To examine this question we conducted a one-way ANOVA for the CS1 Fall 2014 Lab, the CS1 Spring 2016 Project, and the CS2 Fall 2015 Lab. We found that gender had no statistically significant impact on grades or answers to the core survey statements for all three surveys, $p > .05$. For the KenKen (37 students) and Bookstore (27 students) labs, a MANOVA was conducted and showed no statistical association between grades and core survey statements considering both gender and lab title, $p > .05$. Our preliminary analysis of our CS2 Spring 2016 Lab supports this result as well.

Experience: We are also interested in establishing whether and to what degree prior experience in computer science influences grades and responses to the core survey statements. Similarly, we conducted a one-way ANOVA, to investigate any relationship. In our analysis of the first Pencil Puzzle lab in our study, CS1 Fall 2014 Lab: Blackout, we found

Response	CS1-F14	CS1-S16	CS2-B	CS2-K
S: Enjoy	29	39.8	70.3	62.2
S: Frustrating	72.2	40.9	14.8	27
C: Positive	5	9.7	14.8	18.9
C: Negative	21.2	4.3	0	10.8
S: Learn	41.5	52.7	70.3	75.7
S: BelieveICan	43.2	61.3	92.3	81.1
S: CodingSkills	60.3	75.3	62.9	77.8
C: Balanced	12.9	12.9	0	5.4
C: Metacognitive	14.7	12.9	7.4	10.8

Table 3: Responses to Likert-scale survey statements (S) and frequency of comment codes (C) for CS1 in Fall 2014 and Spring 2016, and CS2 in Fall 2015 (Bookstore: -B, Kenken: -K). S rows: % of students who answered "agree" or "strongly agree" to the given statement; C rows: % of students with comments given the particular comment code.

that prior experience in computer science was strongly correlated with all the core survey statements and all reported grades. All p values for the following analyses were below the α level, $p < .05$. The more self-reported experience a student had the more they enjoyed the lab, believed they could code to solve similar problems, did not find the lab frustrating, felt the lab helped them learn, etc. Additionally, lab grade and final grade were strongly correlated to higher experience levels, with higher experience level corresponding to higher grades. Based on feedback from our survey, Blackout as a lab, in CS1 Fall 2014, was overhauled and reworked into a Project. Though this version of the assignment was more challenging, it also was given later in the course and students were given more time to complete it. Our analysis of the most recent iteration of this project, in the CS1 Spring 2016 course, indicated no statistically significant effect of experience on project or final grade.

Our analysis of the parallel sections for CS2 Fall 2015 Lab, using a MANOVA, found that experience, given which lab was completed, only influenced how students rated the lab as helping them improve their coding skills in Java, $p < .014$. An examination of the means plots indicates that across experience levels, KenKen was rated higher as helping students improve their coding skills in Java, while Bookstore, was rated lower, particularly by those with prior experience, at helping them improve their coding skills.

Perception: Putting together data from the core survey statements and the students' voluntary comments, we can form a substantial sense of students' perceptions of the labs and projects. For the purpose of this analysis, we describe perceptions using the responses to the core survey statements Enjoy and Frustrated (see Table 1) and comment codes Positive and Negative (see Table 2). We look to frequency data from the descriptive statistics to report students' perceptions and make comparisons, see Table 3.

Looking to the frequency of responses for the CS1 Fall 2014 Lab we can see students agreed that the Blackout Lab was frustrating, a significant amount of student comments about the Blackout Lab were negative. In contrast a larger percentage of students agreed that they enjoyed the Blackout Project (CS1 Spring 2016), while many fewer reported frustration or made negative comments.

Turning to look closely at our parallel sections, we begin by considering in what ways students perceived Bookstore

differently than KenKen. Students who completed KenKen found it more frustrating, and slightly less enjoyable when compared to how students who completed Bookstore felt about Bookstore. Interestingly, KenKen students volunteered both more positive comments than Bookstore students and many more negative comments than Bookstore students.

Reflection: We are interested to see students' reflective thinking about their own learning in the context of the pencil puzzle assignments. Using responses to survey statements Learn, BelieveICan and CodingSkills and comments coded Balanced and Metacognitive, we attempt to characterize students' reflection (see Tables 1 and 2). Balanced comments were coded as reflective because they show a more complicated thought process by contemplating how something can be in some ways positive and other ways negative. Metacognitive comments illustrate reflection on the lab/project in the context of the student's own learning. For example, the following comment was coded as both balanced and metacognitive: "The lab was difficult for just being a lab assignment but it did help in improving coding skills and understanding of inheritance. The difficult part was easily resolved after spending time reading more into the java docs and the write up."

When we compared Blackout as a lab, CS1 Fall 2014, to the project iteration in CS1 Spring 2016, we found that students made similar amounts of metacognitive and balanced comments, but students in CS1 Spring 2016 more strongly agreed that the Blackout project helped them learn the course material, that they believed they could solve similar puzzles, and that their coding skills improved because of the project, than did students who completed the earlier iteration of Blackout (see Table 3). Turning to our analysis of the CS2 Fall 2015 parallel sections, we recall that in terms of perception, students felt that KenKen was a more frustrating experience, worthy of more negative comments and lower levels of reported enjoyment. Despite these perceptions, more students agreed that KenKen helped them learn the week's material as well as improve their coding skills than did Bookstore students about Bookstore. Additionally, both balanced and metacognitive comments were made more often about KenKen than Bookstore (see Table 3).

4. CONCLUSIONS AND FUTURE WORK

Throughout the duration of this work, we have found pencil puzzles to be a useful context for introductory CS topics in the sense of being able to generate new assignments for a wide variety of different topics. From the student perspective, even when students are frustrated by a challenging puzzle-based assignment, they still often saw the value in it. We also found no association between gender and grades, student perceptions, or student reflective thinking. According to our data pencil puzzle based assignments can be effective in teaching students of varying experience levels. This indicates that pencil puzzles are indeed a leveling context for the instruction of CS topics. In the near future, we will be doing a more thorough analyses of the comments, instructor interviews, and teaching assistant/student lab instructor focus group data to understand in more depth the impact of the pencil puzzle domain on teaching students of all backgrounds as effectively as possible, and will continue to expand our repository to enable any instructor to use this material.

Acknowledgments This material is based upon work supported by the National Science Foundation under grant DUE-1245349. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors would like to thank Don Blaheta for the inspiration for the KenKen assignment, and all the other instructors, TAs, and students in CS1 and CS2 for their cooperation and enthusiasm in assisting with this research.

5. REFERENCES

[1] R. Bates, J. Goldsmith, R. Berne, V. Summet, and N. Veilleux. Science fiction in computer science education. In *SIGCSE '12*, pages 161–162. ACM, 2012.

[2] I. Bezáková, J. E. Heliotis, and S. Strout. On the efficacy of board game strategy development as a first-year CS project. In *SIGCSE '14*, pages 283–288. ACM, 2014.

[3] H. Chang and J. Gibson. The odd-even effect in Sudoku puzzles: effects of working memory, aging, and experience. *Am J Psychol*, 124(3):313–24, 2011.

[4] Z. Dodds, R. Libeskind-Hadas, and E. Bush. When CS 1 is biology 1: crossdisciplinary collaboration as CS context. In *ITiCSE '10*, pages 219–223. ACM, 2010.

[5] P. Drake and K. Sung. Teaching introductory programming with popular board games. In *SIGCSE '11*, pages 619–624. ACM, 2011.

[6] N. Falkner, R. Sooriamurthi, and Z. Michalewicz. Puzzle-based learning for engineering and computer science. pages 20–28, 2010.

[7] C. E. Hmelo-Silver. Problem-based learning: What and how do students learn? *Educational Psychology Review*, 16(3):235–66, September 2004.

[8] Z. Kurmas and J. Vanderhyde. Board game project ideas for CS 1 and CS 2 (abstract only). In *SIGCSE '12*, pages 658–658. ACM, 2012.

[9] Z. Michalewicz and M. Michalewicz. *Puzzle-Based Learning: An introduction to critical thinking, mathematics, and problem solving.* Hybrid Publishers, 2008.

[10] H. Okagbue, M. Adamu, A. Opanuga, Z. Omogbadegun, and E. Obasi. Popularity and gender differences in solving Sudoku game among some sampled secondary school students in Lagos, nigeria. *International Journal of Soft Computing*, 10(6):405–407, 2015.

[11] O. L. Oliveira, A. M. Monteiro, and N. T. Roman. From concrete to abstract?: problem domain in the learning of introductory programming. In *ITiCSE '11*, pages 173–177. ACM, 2011.

[12] Puzzle picnic. `http://www.puzzlepicnic.com/stats`.

[13] Puzzles for introductory computer science. `http://www.cs.rit.edu/~pencilpuzzle`.

[14] S. Ramnath and J. H. Hoover. Enhancing engagement by blending rigor and relevance. In *SIGCSE '16*, pages 108–113. ACM, 2016.

[15] B. Rodriguez, C. Rader, and T. Camp. Using student performance to assess cs unplugged activities in a classroom environment. In *ITiCSE '16*, pages 95–100. ACM, 2016.

[16] I. Russell, K. H. Jin, and M. Sabin. Make and learn: A cs principles course based on the arduino platform. In *ITiCSE '16*, pages 366–366. ACM, 2016.

[17] S. Sheth, C. Murphy, K. A. Ross, and D. Shasha. A course on programming and problem solving. In *SIGCSE '16*, pages 323–328. ACM, 2016.

[18] B. Simon, P. Kinnunen, L. Porter, and D. Zazkis. Experience report: CS1 for majors with media computation. In *ITiCSE '10*, pages 214–218. ACM, 2010.

[19] R. Sooriamurthi, N. Falkner, and Z. Michalewicz. Puzzle-based learning: introducing critical thinking and problem solving for computer science and engineering (abstract only). In *SIGCSE '12*, pages 663–663. ACM, 2012.