Using a Functional Board Game Language to Teach Middle School Programming

Jennifer Parham-Mocello
School of EECS
Oregon State University
Corvallis, USA
parhammj@oregonstate.edu

Martin Erwig
School of EECS
Oregon State University
Corvallis, USA
erwig@oregonstate.edu

Margaret Niess
College of Education
Oregon State University
Corvallis, USA
niessm@oregonstate.edu

Aiden Nelson
School of EECS
Oregon State University
Corvallis, USA
nelsonai@oregonstate.edu

Jason Weber
School of EECS
Oregon State University
Corvallis, USA
webejaso@oregonstate.edu

Garrett Berliner
School of EECS
Oregon State University
Corvallis, USA
berlineg@oregonstate.edu

***Abstract*—In this work (a full paper on an innovative practice), we report on middle school students' experiences while learning a new text-based, functional domain-specific teaching language for programming well-known, simple physical games, such as tossing a coin to see who goes first or playing Tic-Tac-Toe. Based on students' responses after taking an 18-week, 7th grade elective, we find that the majority of the students like learning the new language because it is not block-based, it is not complicated, and it is in the domain of games. However, we also find that there are some students who say programming is what they like the least about the class, and the majority of the students report that they struggle the most with writing the syntax. Overall, the majority of students like the curriculum, language, and using games as a way to explain CS concepts and teach programming. Even though learning a text-based, functional programming language may be difficult for middle-school students, these results show that the domain-specific teaching language is an effective teaching vehicle at the middle school level.**

***Index Terms*—computer science, domain-specific languages, middle-school education, programming**

I. Introduction

We developed a curriculum for introducing computer science (CS) based on identifying computing concepts in simple non-electronic games, which we refer to as the ChildsPlay approach. Our approach is similar to the approaches taken in CS For Fun (CS4FN), the Teaching London Computing, and CTArcade [1]–[3], which also employ physical games to teach CS concepts, but it differs in a fundamental way: Instead of focusing on the strategy for winning games or playing against the computer, we use games without the use of a computer as a model to help students understand CS concepts, such as representation, algorithm, and computation before introducing programming (see Figure 1).

One major goal of the ChildsPlay approach is to debunk negative perceptions of CS by demonstrating to students that understanding basic concepts of computer science is as easy as playing games. We believe choosing games that are well-known for being simple, such as Nim or Tic-Tac-Toe, makes CS more widely accessible for teachers and students.

This work is supported by the National Science Foundation, #1923628.

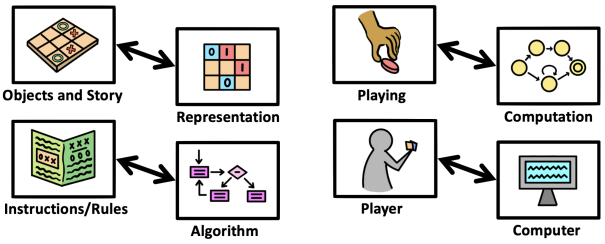

Fig. 1. Connections made between physical games and CS

In this innovative practice report, we discuss the programming curriculum in the ChildsPlay approach, which introduces a new text-based, functional programming language called *BoGL* (short for Board Game Language) after introducing CS concepts using simple, non-electronic games. A current 8th grade mathematics teacher pilots the programming curriculum in a new 7th grade CS elective, and we use the students' assessment and survey responses as insights into the students' experiences with this new curriculum and understanding of the language. More specifically, we address the following two research questions.

1) What are students' understandings about the BoGL syntax for user-defined types, functions, and if-then-else expressions after learning about them in the programming curriculum?
2) What do students like, dislike, learn, and struggle with in the programming curriculum?

The rest of the paper is structured as follows. First, we describe the motivation and related work behind developing the programming curriculum of the ChildsPlay approach in Section II. Then, we describe the curriculum in more detail in Section III, and we report on how the teachers implemented

the curriculum and the data collected in Section IV. In Section V, we provide the student understandings and experiences with the curriculum, and finally, we present conclusions and directions for future work in Section VI.

II. Motivation and Related Work

Playing games helps develop problem-solving skills and creativity, which are fundamental to computational thinking [4]–[6]. Thus, it is not surprising that games have a long tradition as learning tools in education, especially in the form of gamification, which is the idea of representing a learning process as playing a game [7]. While studies have shown that playing board games improves math skills in elementary school students [8] and involves computational thinking activities [9]–[12], simply playing games does not increase one's computational thinking skills, unless guided instruction about the skills is given [13]. Our approach goes beyond just playing games by teaching core CS concepts using simple, physical games for explaining computation.

Several new board and card games have been invented specifically to teach computational thinking (CT), such as RaBit EscAPE (ages 6-10), Cubetto (ages 3-6), and Crabs and Turtles (ages 8-9) [14]–[16], but new games present two disadvantages. First, the rules in new games might not be simple enough and create unnecessary extraneous cognitive load on the learner, taking away cognitive resources from the learning of the computational concepts. Second, schools, kids, and families might not have access to the new games. We believe using existing physical games well-known for being simple broadens participation and shifts the focus to the computational concepts being taught.

The idea of using simple, physical games to explain computational concepts is not new [3], [17], [18], and researchers understand that playing games unsupported by an appropriate framework may be ineffective at teaching the computational concepts [13]. Researchers in the CS4FN and Teaching London Computing projects have shown that the use of games with well-developed lesson plans are effective for teaching specific computational concepts [1], [2], and Lee et al. have shown that their educational software called CTArcade enables children to articulate CT-related thinking patterns while playing Tic-Tac-Toe and Connect Four [3].

We do not use CTArcade, because we want students to play games with their peers to promote social interaction and communication, and we want students and teachers to practice concepts learned in one game by identifying them in other games, which would have to be first implemented in CTArcade. Our approach fits into the landscape of game-based CT teaching approaches by using existing, physical games well-known for being simple, instead of new ones.

CS Unplugged [19], [20] has been shown to broaden participation [21], and several studies have demonstrated that unplugged activities, such as games, puzzles, and storytelling, can be a viable alternative to traditional programming activities for teaching introductory computational skills and algorithms [20], [22]–[28]. Supporting studies have shown the positive impacts unplugged activities have on students' perspectives of, engagement in, and motivation to study CS [27], [29]–[32]. For these reasons, we also avoid the use of technology and programming for introducing CS concepts.

After introducing students to fundamental concepts in CS using physical games in an unplugged environment, students can then apply the concepts to programming. However, Brusilovsky et al. argue that one of the obstacles general-purpose languages pose to beginning students includes being too large and cognitively overwhelming [33]. Likewise, educators have shown success using domain-specific languages for introducing programming [34], [35]. Other researchers argue that the reduced complexity and natural relationship to familiar mathematical concepts, in addition to leveling the playing field, makes functional languages a better choice for teachers and students [36]–[41], which is what spearheaded the successful Bootstrap Algebra project [42], [43]. For these reasons, we think it is important to use a functional, domain-specific language as the beginning language for students. While we understand that block-based languages have been shown to help students understand some programming concepts better [44], [45], in this project, we are specifically interested in introducing a text-based language for expressing algorithms in a formal notation.

III. Curriculum Background

The researchers and CS students developed the programming curriculum in collaboration with two teachers from a middle school in the United States. We briefly summarize the researcher-practitioner partnership and the curriculum developed and piloted at the local middle school during the 2020/2021 academic year, as a result of this partnership.

A. Developing the Curriculum: A Researcher Practitioner Partnership

Building on a well-established collaboration with a local dual-language immersion middle school, we developed a programming curriculum that gradually introduces CS by using simple, physical games and a domain-specific language to teach fundamental concepts. The researchers worked with two mathematics teachers, as well as the Assistant Principal, for 1-2 hours each month throughout the previous 2019/2020 school year building the teachers' understanding of fundamental CS concepts. One teacher was a 6th grade mathematics teacher with a BS in primary education, and she was in her first year of teaching during the 2019/2020 development phase. The other teacher was an 8th grade mathematics teacher with a MS in secondary education, and he was in his sixth year of teaching during the development phase. Neither teacher had a background in CS or prior programming experience.

Before offering the 7th-grade programming elective, the researchers engaged with the practitioners in an iterative process to polish the curriculum and teachers' knowledge for delivering the CS content. First, the teachers participated in a summer workshop for 40 hours over two weeks addressing questions and teaching concepts related to the curriculum. This

training was critical for the development of the teachers' CS content knowledge. We revised the curriculum based off of the information and feedback we gathered from the workshop. Then, the teachers used the curriculum to teach middle-school students in a 1-week, 3-hours-per-day online summer camp to help develop the teachers' technical pedagogical content knowledge (TPACK) [46] for teaching CS, which was especially important due to the shift to being online because of COVID-19. The one-on-one training activities and summer camps led to the final curriculum and the teachers' TPACK for delivering the electives in the 2020/2021 academic year.

B. The Resulting Programming Curriculum: A Domain-Specific Teaching Language

The goal of the programming curriculum is to introduce a formal notation for algorithms within the scope of board games. To this end, we designed BoGL (the Board Game Language) [47]. BoGL has a web interface to make it accessible by anyone on any platform with internet access (see Figure 2). The curriculum covers approximately 60 hours of instruction over 18 weeks.

BoGL is a domain-specific teaching language (DSTL) targeted at writing programs in the domain of board games but used to easily move students to a general-purpose language. BoGL is primarily a functional, text-based language syntactically similar to Haskell [48] or Elm [49], but with a significantly simplified syntax and type system.

Mirroring the non-programming curriculum, which begins with the concept of representation, the programming curriculum introduces types and values before functions, which are the formal equivalent of algorithms in BoGL. We start with simple functions to illustrate the use of parameters, followed by simple expressions, control structures, and conditions. The curriculum ends with introducing repetition and the concept of an array data structure to represent game boards.

We also support the smooth transition from algorithmic notation to a BoGL program through a manual process we call *BoGLization*, which shows an algorithm and the development of the BoGL program side by side, highlighting corresponding parts in both. An example is shown in Figure 3. Referring to Figure 3, the partial BoGL program (shown on top) is manually created by the teacher from an algorithm (shown at the bottom) in several steps. Parts in the algorithm that have been translated in previous steps are shown in gray. The light blue background focuses on those parts in the algorithm that are translated into BoGL in the current step. Currently, this translation is not automated; it is a manual process that the teacher goes through with the students to build their understanding of how to go from an algorithm description to a formal programming language. We are currently investigating how to support this process through an interactive tool.

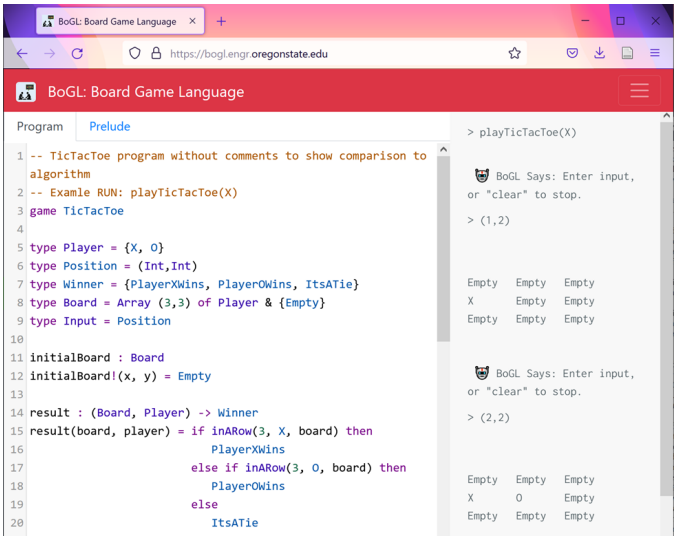

Fig. 2. Screenshot of a Tic-Tac-Toe program in BoGL.

IV. Research Method

In this section, we present the research method used to determine students' understanding of and experiences with our curriculum. We first describe the implementation of the 7th grade elective, and then we describe the data collected to answer our research questions.

A. The 7th grade CS Elective

Because 2020/2021 was the first year offering the curriculum, students did not get exposure to the non-programming 6th-grade curriculum in the prior year, the teacher covered most of the non-programming curriculum in the first 3-4 weeks of the 18 weeks. Then, the students used BoGL to reinforce the concept of how algorithms support the design of programs and the concepts of types, names, values, functions, and control structures with conditions.

Since the school was online due to COVID-19, the teachers used Zoom for class lectures and Canvas to organize all the information for the class, such as the syllabus, grades, etc. The teachers used Kahoot! [50] games as brain breaks for the students, and the students played physical board games, like Boggle, Connect Four, or SET, as a class or in a Zoom breakout room online each week.

The teachers had access to the pre-designed lesson plans, presentation slides, and student worksheets. The teachers were free to modify activities and material, add new material, or

```
game CoinToss

type Coin = {Heads,Tails}
type Child = {Jack,Rosa}
```

ALGORITHM Coin Toss
Jack picks heads or tails for the winning side
Rosa tosses the coin
IF the coin toss is equal to the winning side THEN
    Rosa does the dishes
ELSE
    Jack does the dishes

Fig. 3. BoGLization of the Coin Toss algorithm.

strictly keep to the lesson plans as presented. The 7th grade elective teacher did not modify the provided curriculum before the first elective offering. By the second offering, we noticed the teacher felt comfortable making some modifications by adding "skeleton" code with fill in the blanks for the students, which he felt was easier for them to understand than creating a program from a blank page. However, he never felt comfortable enough creating entirely new programs or assignments for the students.

1) Student Background: Approximately, 28-30 students were in each semester elective, which was only about 60 students total. With IRB approval, 37 students (25 identifying as male and 12 identifying as female) consented to participate in the research study and took the pre-survey. However, only 30 students (20 male and 10 female) shared their experiences in the post-survey.

Most of the students self-selected to be in the elective, but there were 10 students (8 in the post-survey) who did not choose to take the elective. This is due to the school reducing the number of electives offered online due to COVID-19, and the school moved some students into other electives. While 26 of the 37 students reported having prior programming experience, only 16 students reported attending a prior class or camp for programming or CS, which could be because students did not consider coding/CT activities, such as an "Hour of Code" or self-taught exercises, as a class.

2) Assessment Questions: At the end of the units on functions and if-then-else, we provided exit tickets to assess student understanding/recognition of the BoGL syntax and tracing algorithms (see Figure 4). The exit ticket on types and functions was multiple choice with multiple correct answers, and the second exit ticket on tracing algorithms and if-then-else contained free-response and multiple choice with one and multiple correct answers. At the end of the elective, we provided a post survey with five open-ended questions asking students about their likes, dislikes, struggles, and learning in the elective, as well as what they thought about BoGL and using games to teach programming.

Multiple Choice

Question	Answers (multiple select)
1. Use the following type definition: `type NumPair = (Int, Int)` Which function signatures are valid in BoGL?	• fun : type NumPair -> type NumPair • fun : Int-> NumPair • fun : NumPair -> Int • fun : NumPair -> Int, Int • fun : (Int, Int) -> NumPair

2. Use the following type definition:
`type NumPair = (Int, Int)`
Given the function signature below, which function definitions are valid in BoGL?
`fun : NumPair -> NumPair`
	• fun(a, b) = ((a+b), (a-b))
	• fun a, b = ((a-a), (b-b))
	• fun(a, b) = ((a+b+5), (5))
	• fun(a, b) = a, b

3. Use the following type definitions:
`type NumPair = (Int, Int)`
`type Bag = (NumPair, NumPair)`
Given the function signature below, which function definitions are valid in BoGL?
`swap : Bag -> Bag`
	• swap(p, k) = k, p
	• swap(p, k) = (k, p)
	• swap(p, k) = (k p)
	• swap(p k) = (k, p)

Algorithm Analysis

Algorithm #1

Use the following function signature and definition to answer the following questions:

```
1| foo : Int -> Bool
2| foo (num) = if num > 0 then True
3|              else False
```

Question	Answers
4. When would this function output False?	*Free Response* (when num is <= 0)
5. If you run/execute the function with "`foo(0)`", what would it output?	• True • False

Algorithm #2

Use the following function signature and definition to answer the following questions:

```
1| type Direction = {Left, Right, Up, Down}
2|
3| movePlayer : (Int, Int, Direction) -> (Int, Int)
4| movePlayer(x, y, direction) = if direction == Left then (x-1, y)
5|    else if direction == Right then (x+1, y)
6|    else if direction == Up then (x, y+1)
7|    else (x, y-1)
```

Question	Answers
6. What does the function "movePlayer" output when the function is run/executed using "`movePlayer(3, 7, Down)`"?	• (3, 6) • (3, 8) • (2, 7) • (4, 7)
7. When would this function run/execute "`then (x+1, y)`" on line 5?	*Free Response* (when direction is (or equal to) Right)

Algorithm #3

Use the following function signature and definition to answer the following question:

```
1| fun : (Int, Int) -> (Int, Int)
2| fun(a, b) = ((a+b), (a-b))
```

Question	Answers (multiple select)
8. Which of the following are valid ways to run/execute the "fun" function in BoGL?	• fun(1, 2) • fun(4) • fun((3+1), (5+9))

Fig. 4. Multiple Choice and Algorithm Analysis Questions

V. Results

In this section, we present the experiences of students in the 7th grade elective by analyzing their multiple-choice and free-response answers to exit tickets and categorizing their qualitative responses to post-survey questions into themes. For some post-survey question responses, we categorized them into more than one theme. We counted the number of times each theme appears in the student responses to quantitatively evaluate overall experiences with the curriculum and the new functional, text-based DSTL.

A. Student Understanding About BoGL Syntax

In this section, we present how students understand the syntax of functions and if-then-else in BoGL. Only 19 students from the first semester took the first exit ticket on function syntax (see Figure 5), and 27 students (19 from the first semester and 8 from the second semester) took the second exit ticket on tracing algorithms (see Figure 6). Unfortunately, the researchers forgot to remind the teacher about the first exit ticket in the second semester, and the teacher forgot to give the first exit ticket to second-semester students. The researchers and teacher remembered the second exit ticket.

1) What are students' understandings of the function syntax in BoGL?: In the first question, we asked students to identify valid function signatures in BoGL, given a new type called NumPair that is a pair of Ints (see Figure 4). Most students (16 out of 19) recognized that the keyword "type" was never used in a function signature, and the majority of the students recognized that the last signature with parentheses around the pair of Ints was correct (see Figure 5). However, out of 14 students who correctly recognized the pair of Ints with a parentheses, five also incorrectly selected the signature without parentheses around the pair of Ints. Interestingly, only four students selected the signature with an Int as input and a NumPair as output. Whereas, seven students selected the signature with a NumPair as input and an Int as output. In either case, the number of students selecting options with one

Participant	Question #1					Question #2				Question #3		
	Excludes "type" answer	Includes answer, fun:Int->NumPair	Includes answer, fun:NumPair -> Int	Includes answer, fun:(Int, Int) -> NumPair	Includes answer and not fun: NumPair -> Int, Int	Excludes fun a, b	Includes answer, fun(a, b) = ((a+b), (a-b))	Includes answer, fun(a,b) =(a+b+5), (5))	Excludes fun(a,b) = a,b	Excludes "missing comma" answers	Excludes "missing parens" answer	Includes correct answer
11	1	1	1	1	1	1	1	1	1	1	1	1
26	1	1	1	1	1	1	1	0	0	1	1	1
18	1	0	1	1	0	1	1	1	1	1	0	1
10	1	0	0	1	0	1	1	0	0	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1
22	1	0	0	1	0	1	1	0	0	1	1	1
17	1	0	0	1	1	1	1	0	1	1	1	1
9	1	0	0	0	0	0	0	0	1	1	1	1
14	0	1	1	0	0	1	1	1	0	1	1	1
25	1	0	0	1	1	1	0	0	0	1	0	0
7	1	0	0	0	0	1	1	0	1	1	1	1
19	0	0	0	0	0	1	1	0	1	1	1	1
20	1	0	1	1	1	0	0	0	0	0	1	1
8	1	0	0	1	0	0	1	0	1	0	1	1
16	0	0	0	0	0	0	1	1	1	1	1	1
12	1	0	1	1	1	1	0	0	0	1	1	1
1	1	0	0	1	0	1	1	0	1	1	1	1
24	1	0	0	1	1	1	1	0	1	1	1	1
23	1	0	0	1	1	0	0	0	1	0	1	1

Fig. 5. Answers to Multiple Choice Function Syntax Questions

Participant	Algorithm #1			Algorithm #2			Algorithm #3		
	Question #4		Q #5	Q #6	Question #7		Question #8		
	Includes < 0	Includes == 0	foo(0) is False	correct answer (3, 6)	Right not direction	direction is Right	Excludes fun(4) answer	Includes answer, fun(1,2)	Includes answer, fun((3+1), (5+9))
11	1	0	1	1	1	1	1	1	0
26	1	0	1	1	1	1	1	1	0
18	1	0	1	0	0	0	1	1	0
10	1	0	1	1	1	0	1	1	0
5	1	1	1	1	1	1	1	1	0
22	0	0	1	1	0	0	1	1	0
17	0	0	1	0	1	1	1	1	0
9	1	0	0	0	1	0	1	1	0
14	0	1	1	1	1	1	1	1	1
25	0	1	1	1	1	1	0	1	0
7	1	0	0	0	1	0	1	1	0
19	0	1	1	1	0	0	1	1	0
8	1	0	1	1	1	1	1	0	1
16	1	0	1	0	1	1	1	1	1
12	1	0	1	1	0	0	1	1	1
1	1	1	0	0	0	0	1	1	0
24	1	0	1	1	1	1	1	1	0
2	0	0	1	0	0	0	1	1	1
13	1	1	1	1	1	1	1	1	1
66	0	0	0	0	0	0	1	1	0
69	0	0	0	0	1	1	1	1	0
60	1	1	0	1	1	1	1	0	1
65	0	0	1	0	0	0	0	0	0
62	0	0	1	1	1	0	0	1	0
64	1	0	1	1	1	1	1	1	0
63	1	0	1	0	1	1	1	0	1
68	0	1	0	0	0	0	1	1	0

Fig. 6. Answers to Algorithm Analysis Questions

Int as input or output with a NumPair was low.

In the second and third questions, we asked students to identify a valid function definition, given the same NumPair type definition with an additional Bag type with two NumPairs in Question #3 and a function signature (see Figure 4). We wanted to further evaluate students' understanding of tuple types and the use of parentheses.

As suspected, the majority of the students (14 out of 19) correctly recognized missing parentheses around the parameters beside the name of the defined function in Question #2, but most of the students (6 out of 7) who incorrectly chose the definition without parentheses around the pair of Ints being returned were also those who correctly recognized missing parentheses around the parameters (see Figure 5). Figure 5 also shows that most students (14 out of 19) correctly identified the most obviously correct function definition, but most students did not choose the other answer with the extra parentheses around the single literal integer value in the second pair of integers, which was a tricky question.

In Question #3, most students recognized that a function definition cannot have missing commas between the parameter names or returned output (see Figures 4 and 5). Also, interestingly, the majority of students (17 out of 19) did not incorrectly choose the option with missing parentheses around the returned pair of integers in Question #3, as they did with the last option in Question #2 (see Figure 4).

In the last question of the second exit ticket, we asked students a general question about possible executions of a function given a specific function definition (see Figure 4). We were looking for students to recognize that there were two inputs to this function. Therefore, the function needed to be applied to two values. Most students (24 out of 27) understood this concept and did not select the incorrect option with only one input value (see Figure 6). However, three students, who did not include the option with only one input, did not include the correct answer with two literal inputs, and only selected the option with two math expressions as input. Most students (19 out of 27) did not select the option with the math expressions and parentheses for the two input values, but this was not surprising based on student responses in the first exit ticket.

2) What are students' understandings of tracing functions with if-then-else in BoGL?: In the first four questions of the second exit ticket, we asked students to identify properties about the if-then-else conditions and to perform a trace (or computation) with given inputs to functions with if-then-else (see the "Algorithm Analysis" portion of Figure 4). In the first algorithm question, we asked students to explain what would make the function output "False". We were looking for students to say something about being less than and equal to zero. While many students (16 out of 27) said something about being less than zero, most students (19 out of 27) did not include being equal to zero in their response (see Figure 6). It is interesting that four students mentioned when the number was zero without mentioning being less than zero.

In Question #5, we asked students to recognize what the output would be when the input was zero (see Figure 4), and most students (20 out of 27) correctly selected "False" (see Figure 6). This might suggest that 7th grade students can better recognize a correct answer than provide all pieces to a correct answer in an open-ended question. However, it was surprising that approximately 30% of the students got this simple tracing exercise wrong.

Next, we asked students to trace another if-then-else with specific input (see Figure 4), and even though it was a multiple choice question with only one option to choose, many students (12 out of 27) struggled with the question (see Figure 6). When the direction is "Down", then the output should be the same x value and one subtracted from the y value. However, the majority of the students (8 out of the 12) either increased the

y value or increased the x value, which was unexpected from the middle-school math learned.

With a more concrete Question #7 that asked students to explain when the x value was increased and the y value stayed the same (see Figure 4), only 9 out of the 27 students did not mention something about "Right", and only 4 out of 18 students, who mentioned "Right", did not say when the direction was equal to "Right" (see Figure 6), which was different than student open-ended responses to Question #4. In any case, 30% or more students struggled with analyzing and tracing algorithms with if-then-else control structures.

B. Student Experiences

In this section, we present the experiences of the 30 assenting students who took the post-survey in the 7th grade elective by categorizing their qualitative post-survey responses into themes. Some student responses were categorized into more than one theme, and for each survey question, we counted the number of times each theme appeared in a student response to quantitatively evaluate their overall experiences with the curriculum.

1) What do students like the most about the elective?: First, we asked students what they liked about the class. The majority of students mentioned something about coding, using BoGL, programming games, or getting their code to work as what they like the most in the elective (see Figure 7). These responses were not surprising in an elective. However, out of the 8 students who did not choose the elective, three of the students mentioned liking coding, and one liked learning something new. The other four, who did not choose to take the elective, liked working in teams, playing Kahoot! games, the teacher, and the energy of the class. It was interesting that 5-6 students specifically mentioned the teacher and teams as something they liked about the class, and many students mentioned only one or both of these aspects as what they liked the most in the class.

2) What do students like the least about the elective?: Just as we asked students about what they liked, we also asked them what they did not like. Many students (10 out of 30) said that there was nothing they liked the least about the class (see Figure 7). However, four students specifically said something about the class being on Zoom or using breakout rooms. Four students explicitly stated not liking coding, and two students mentioned not liking debugging their code or being confused by code. However, only three of the seven who mentioned something negative about coding did not choose to take the elective. Whereas, two of the three who mentioned something about the class being hard or having a lot of material also did not choose to take the class. Other students mentioned not liking either the non-programming part of the class, remembering to insert screenshots in their assignments, the lectures on input and output to functions, specifics about the syntax, waiting to start programming, or specific games.

3) What do students feel they learn from the elective?: Even though the curriculum started with teaching fundamental CS concepts without programming, the majority of the students (27 out of 30) felt they learned about coding the most in the class (see Figure 7), which is not surprising when they spent 14-15 weeks (out of 18 weeks) on programming. It was interesting that two students explicitly mentioned types and functions as what they learned in programming. Although only two students reported that their beliefs about CS and programming were changed, their conclusions were exactly in line with two of the major goals of the curriculum, namely:

(A) illustrating that programming is not intimidating, and
(B) providing students with an accurate view of CS.

We believe that goal (A) is particularly relevant for under-represented groups in CS and those not self-selecting CS, and both of these criteria apply to the student who reported that change in belief. Goal (B) is important to provide clarity to students who think they want to do CS because they play video games or it is popular.

4) What do students feel they struggle with the most in the elective?: While many students said coding was what they liked most about the class and learned in the class, the majority of students (19 out of 30) also stated that writing the code and the details of the syntax was what they struggled with the most in the elective (see Figure 7). Three students mentioned

Theme	Responses
Like the most	
Coding/BoGL/Program Games/Debugging	11
Teacher	6
Teams	5
Everything	4
Class: Pace/Energy/Relatable/Applicable	4
Kahoots/Game Day	3
Vocabulary/Computing	2
Something New	2
Learn from the class	
Code/BoGL/text-based/syntax/debugging	27
Types and Functions	2
Code is not intimidating	1
CS is complicated and requires practice	1
Like the least	
Nothing	10
Class: Non-programming/Assignments/IO/ Syntax/Waiting	5
Class on Zoom/Breakout Rooms	4
Coding	4
Hard Class/Lots of Material	3
Buggy Code/Being Confused	2
Specific Games: RPS and Nim	2
Struggle with the most	
Writing the code with all the details/syntax.	19
Understanding code	3
Functions and Types	3
Debugging: Find/Fix errors	2
Non-programming part	2
Writing Nim	1
Running the program	1

Fig. 7. Student Experiences in the 7th-grade Elective

specifically struggling with functions and types, and five other students mentioned struggling to understand and debug code. It was interesting that two students stated struggling with the non-programming/boring parts of the class, and one also mentioned disliking the non-programming part of the class the most. One of these students had no prior programming experience, while the other reported some experience using Scratch.

5) Do students like using games as a way to learn fundamental CS concepts?: The overwhelming majority of the students (28 out of 30) liked using games as a way to teach CS concepts and programming (see Figure 8). Most students stated that they liked using games because it is fun, the best way to learn, and not boring. One student specifically said that the use of games helped them focus. Two students mentioned that games were cool to program/learn how they work in code and making games was cool to show your family, which we also had as a goal of this curriculum. Since games are usually played with friends and family, we believe that games provide a common language for kids to share what they learn about CS with their friends and family.

6) How well do students enjoy programming in a new functional, text-based language?: Finally, we asked students what they thought about the new programming language used in their class. The majority of students (23 out of 30) liked using BoGL. Four students did not like the language at all, and three students sort of liked the language (see Figure 9).

Students gave many reasons for liking or disliking the new language. We provide a list of the reasons with the number of students who listed the reason. Since one of our goals was to expose students to a text-based language in early middle school, it was good to see that a reason for liking the language was because it was not block-based.

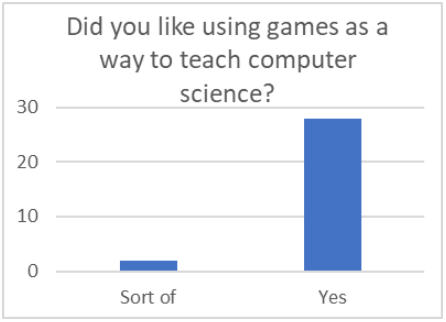

Fig. 8. Student Feelings Toward Games

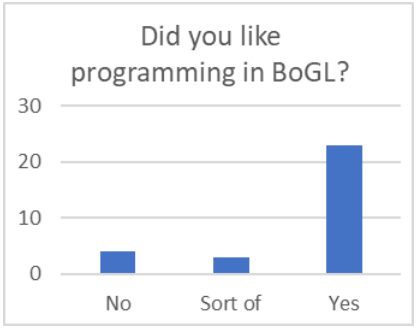

Fig. 9. Student Feelings Toward BoGL

It was interesting that some students explicitly stated liking the language because it was simple, not complicated, and made sense compared to other text-based languages, which was another goal of making it a DSTL. One student goes as far as saying that they tried learning other text-based languages that did not make sense before BoGL.

Only two students did not like the language because they preferred other languages, which did not seem like a primary reason for disliking the language. Other possible reasons for students to dislike the language were because they did not understand programming or like programming at all. Reasons for students sort of liking the language were because of platform issues or general frustration from learning how to program, but they stated otherwise liking the language.

Reasons for liking BoGL

- New coding language they didn't know or it wasn't block based (4)
- Not complicated/simple (3)
- Makes sense (2)
- Rewarding to figure things out/get it working (2)
- Setup/user interface/errors/syntax highlighting (4)
- Engages them in thinking (2)
- Fun/the best (4)

Reasons for disliking BoGL

- Prefer other coding language (2)
- Do not understand/do not like coding (2)

Reasons for sort of liking or disliking BoGL

- Glitchy/platform issues (3)
- Fun but frustrating/difficult (2)

VI. Conclusions

From this experience, we learned that the teacher and class structure contributed significantly to students' positive experiences in the elective. While the majority of the students understood some of the basic syntax and concepts in BoGL, some students struggled with expanding their knowledge to new situations, such as with parentheses for pairing information and mathematical expressions as input values. Students also had a hard time analyzing and tracing algorithms correctly, which were not concepts emphasized in the curriculum. In the future, we will emphasize the concept of computation by integrating more activities on algorithm analysis, tracing, and prediction, which could also help with debugging skills.

However, we did see that the majority of students like the curriculum and using games as a way to explain CS concepts and teach programming. Even though most students self-selected into the elective, the experience was positive for those who did not choose to take the elective on their own.

We were surprised by how much prior programming experience the middle school students had in the 7th grade, but this did not seem to have an impact on their feelings toward the

curriculum or the language. Most students were appreciative of learning a text-based language instead of a block-based language. This might be attributed to the fact that BoGL is a DSTL that is simple to use.

Only few students did not like the non-programming part of the curriculum, and the gentle slope could lead to the positive feeling toward the curriculum and less intimidation with learning how to program, as evidenced by some of the students' responses.

As with any innovative practice report or research study, we recognize that the evaluation of these results were only from half the students in the class; other students might have had different experiences. Therefore, the list of likes and dislikes about the curriculum and new programming language is not exhaustive, but it does give some insights into what students understand about the BoGL syntax and think about the curriculum.

In the future, we plan to use these results to further improve the curriculum. For example, to address students' struggle with syntax, we will provide a gentler slope into learning the syntax of the language by utilizing the interpreter before the editor. Students will write expressions on Integers and Booleans to motivate the idea that operations take certain types of values as input and produce a certain type of value as output. Playing with small expressions and exploring syntax in the interpreter can also gradually accustom students to deal with errors. We also contemplate making minor changes to BoGL, based on students' interaction with the language. Finally, we are currently working on formalizing the process of BoGLization and implementing a tool that can assist teachers and students with translating their algorithm descriptions to BoGL programs.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under the grant #1923628.

REFERENCES

[1] CS For Fun: Queen Mary, University of London, "Welcome to cs4fn : the fun side of computer science," http://www.cs4fn.org/, 2011, accessed: 2021-01-07.

[2] ——, "Teaching london computing: A resource hub from cas london & cs4fn," https://teachinglondoncomputing.org/, 2015, accessed: 2021-01-07.

[3] T. Y. Lee, M. L. Mauriello, J. Ahn, and B. B. Bederson, "Ctarcade: Computational thinking with games in school age children," *Int. Journal of Child-Computer Interaction*, vol. 2, no. 1, pp. 26–33, 2014.

[4] L. A. Sharp, "Stealth Learning: Unexpected Learning Opportunities Through Games," *Journal of Instructional Research*, vol. 1, pp. 42–48, 2012.

[5] C. Harris, "Meet the New School Board: Board Games Are Back–And They're Exactly What Your Curriculum Needs," *School Library Journal*, no. 5, pp. 24–26, 2009.

[6] C. Ragatz and Z. Ragatz, "Tabletop Games in a Digital World," *Parenting for High Potential*, no. 7, pp. 16–19, 2018.

[7] K. M. Kapp, *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education*. Pfeiffer, 2012.

[8] S. Cavanagh, "Playing Games in Class Helps Students Grasp Math," *Education Digest: Essential Readings Condensed for Quick Review*, no. 3, pp. 43–46, 2008.

[9] M. Berland and S. Duncan, "Computational Thinking in the Wild: Uncovering Complex Collaborative Thinking through Gameplay," *Educational Technology*, vol. 56, no. 3, pp. 29–35, 2016.

[10] M. Berland and V. R. Lee, "Collaborative Strategic Board Games as a Site for Distributed Computational Thinking," *Int. Journal of Game-Based Learning*, vol. 1, no. 2, pp. 65–81, 2011.

[11] N. R. Holbert and U. Wilensky, "Racing games for exploring kinematics: a computational thinking approach," 2011, pp. 109–118.

[12] C. Kazimoglu, M. Kiernan, L. Bacon, and L. MacKinnon, "Learning programming at the computational thinking level via digital game-play," *Procedia Computer Science*, vol. 9, pp. 522–531, 2012.

[13] T. Y. Lee, M. L. Mauriello, J. Ingraham, A. Sopan, J. Ahn, and B. B. Bederson, "CTArcade: Learning Computational Thinking Thile Training Virtual Characters Through Game Play," in *Human Factors in Computing Systems*, 2012, pp. 2309–2314.

[14] P. Apostolellis, M. Stewart, C. Frisina, and D. Kafura, "RaBit EscAPE: A Board Game for Computational Thinking." in *Conference on Interaction Design and Children*, 2014, pp. 349–352.

[15] Primo, "Cubetto: Screenless coding toy for girls and boys aged 3-6," 2018, https://www.primotoys.com.

[16] K. Tsarava, K. Moeller, and M. Ninaus, "Training Computational Thinking Through Board Games: The case of Crabs and Turtles," *Int. Journal of Serious Games*, vol. 5, no. 2, pp. 25–44, 2018.

[17] CS For Fun: Queen Mary, University of London, "Winning at nim: computers outwitting humans," http://www.cs4fn.org/binary/nim/nim.php, 2011, accessed: 2021-01-07.

[18] ——, "Noughts & crosses," http://www.cs4fn.org/programming/noughts-crosses, 2011, accessed: 2021-01-07.

[19] T. C. Bell, I. H. Witten, and M. Fellows, *Computer Science Unplugged: Off-line Activities and Games for All Ages*. Computer Science Unplugged, 1998.

[20] T. Bell, I. H. Witten, and M. Fellows, *CS Unplugged. An Enrichment and Extension Programme for Primary-Aged Students*, 2015.

[21] T. J. Cortina, "Reaching a Broader Population of Students Through "Unplugged" Activities," *Communications of the ACM*, vol. 58, no. 3, pp. 25–27, 2015.

[22] R. Thies and J. Vahrenhold, "On Plugging "Unplugged" Into CS Classes," 2013, pp. 365–370.

[23] Q. Cutts, Q. Connor, G. Michaelson, and P. Donaldson, "Code or (not code): separating formal and natural language in CS education," 2014, pp. 20–28.

[24] J. Parham-Mocello, S. Ernst, M. Erwig, E. Dominguez, and L. Shellhammer, "Story Programming: Explaining Computer Science Before Coding," in *ACM SIGCSE Symp. on Computer Science Education*, 2019, pp. 379–385.

[25] J. Parham-Mocello and M. Erwig, "Does Story Programming Prepare for Coding?" in *ACM SIGCSE Symp. on Computer Science Education*, 2020, pp. 100–106.

[26] J. Parham-Mocello, A. Nelson, and M. Erwig, "Exploring the Use of Games and a Domain-Specific Teaching Language in CS0," in *ACM Conf. on Innovation and Technology in Computer Science Education*, 2022, pp. 351—-357.

[27] P. Curzon, P. W. McOwan, N. Plant, and L. R. Meagher, "Introducing teachers to computational thinking using unplugged storytelling." 2014, pp. 89–92.

[28] Primo, "Free beginner's guide to coding with kids," https://www.primotoys.com/guide-coding-for-kids-ebook/, 2020, accessed: 2021-01-07.

[29] T. Bell, P. Curzon, Q. I. Cutts, V. Dagiene, and B. Haberman, "Overcoming Obstacles to CS Education by Using Non-Programming Outreach Programmes," in *Int. Conf. on Informatics in Schools*, ser. LNCS 7013, 2011, pp. 71–81.

[30] Q. I. Cutts, M. I. Brown, L. Kemp, and C. Matheson, "Enthusing and informing potential computer science students and their teachers," in *SIGCSE Conf. on Innovation and Technology in Computer Science*, 2007, pp. 196–200.

[31] C. Mano, V. Allan, and D. Cooley, "Effective In-Class Activities for Middle School Outreach Programs," in *Annual Conf. on Frontiers in Education*, 2010, pp. F2E–1–F2E–6.

[32] R. Taub, M. Ben-Ari, and M. Armoni, "The Effect of CS Unplugged on Middle-School Students' Views of CS," in *SIGCSE Conf. on Innovation and Technology in Computer Science*, 2009, pp. 99–103.

[33] P. Brusilovsky, E. Calabrese, J. Hvorecky, A. Kouchnirenko, and P. Miller, "Mini-languages: a way to learn programming principles,"

Education and Information Technologies, vol. 2, no. 1, pp. 65–83, 1997. [Online]. Available: https://doi.org/10.1023/A:1018636507883

[34] T. Kosar, N. Oliveira, M. Mernik, M. Pereira, Črepinšek, D. D. Cruz, and P. H. Rangel, "Comparing general-purpose and domain-specific languages: An empirical study," *Computer Science and Information Systems*, vol. 7, no. 2, pp. 247–264, 2010.

[35] T. Kosar, M. Mernik, and J. C. Carver, "Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments," *Empirical Software Engineering*, vol. 17, no. 3, pp. 276–304, 2012.

[36] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi, "The structure and interpretation of the computer science curriculum," vol. 14, no. 4, pp. 365–378. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0956796804005076/type/journal_article

[37] ——, "The TeachScheme! project: Computing and programming for every student," *Computer Science Education*, vol. 14, no. 1, pp. 55–77. [Online]. Available: https://www.tandfonline.com/doi/full/10.1076/csed.14.1.55.23499

[38] M. M. T. Chakravarty and G. Keller, "The risks and benefits of teaching purely functional programming in first year," vol. 14, no. 1, pp. 113–123. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0956796803004805/type/journal_article

[39] J. Hughes, "Experiences from teaching functional programming at chalmers," vol. 43, no. 11, pp. 77–80. [Online]. Available: https://doi.org/10.1145/1480828.1480845

[40] J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*. Cambridge, MA: MIT Press, 2003.

[41] S. Joosten, K. Van Den Berg, and G. Van Der Hoeven, "Teaching functional programming to first-year students," vol. 3, no. 1, pp. 49–65. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0956796800000599/type/journal_article

[42] B. Community, "Bootstrap." [Online]. Available: https://bootstrapworld.org/materials/algebra/

[43] G. Wright, P. J. Rich, and R. Lee, "The influence of teaching programming on learning mathematics," 2013.

[44] M. Mladenovi, S. Mladenovi, and Žana Žanko, "Impact of used programming language for k-12 students' understanding of the loop concept," *International Journal of Technology Enhanced Learning*, vol. 12, pp. 79–98, 2020.

[45] N. Humble, "The use of programming tools in teaching and learning material by k-12 teachers," 10 2021.

[46] P. Mishra and M. J. Koehler, "Technological pedagogical content knowledge: A framework for teacher knowledge," *Teachers College Record*, vol. 108, pp. 1017–1054, 2006.

[47] Friedman, B. and Grejuc, A. and Erwig, M., "BoGL Web Implementation," 2020, bogl.engr.oregonstate.edu.

[48] Haskell, "An advanced, purely functional programming language," https://www.haskell.org, 2019, accessed: 2020-08-24.

[49] Elm, "A delightful language for reliable web applications." https://elm-lang.org, 2021, accessed: 2021-08-11.

[50] Kahoot!, "A game-based learning platform," https://kahoot.it/, 2020, accessed: 2020-08-26.