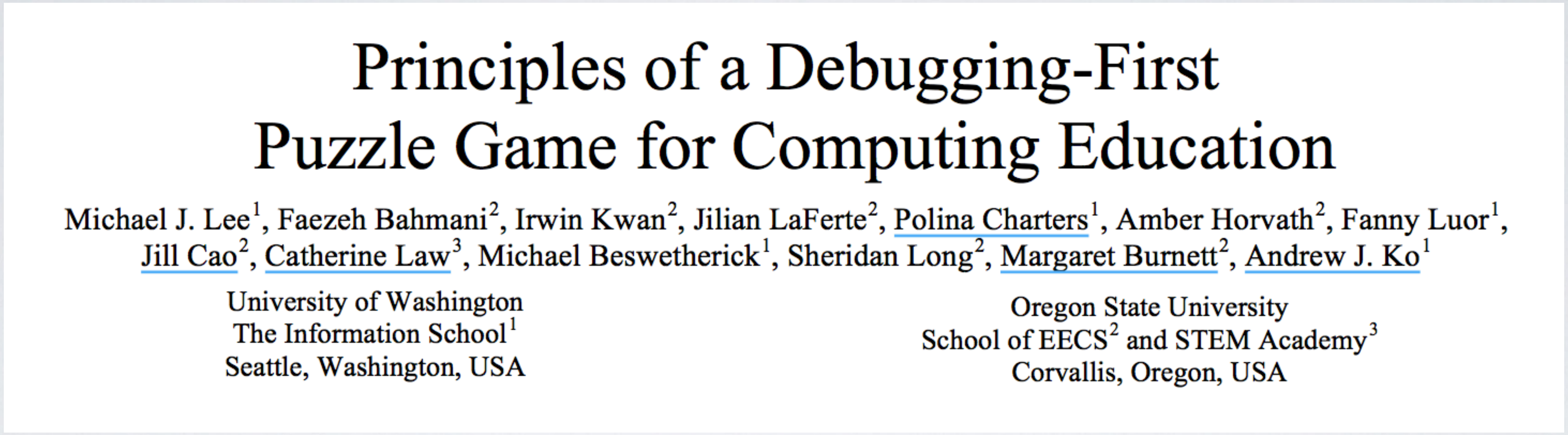


Principles of a Debugging-First Puzzle Game for Computing Education

Michael J. Lee[1], Faezeh Bahmani[2], Irwin Kwan[2], Jilian LaFerte[2], Polina Charters[1], Amber Horvath[2], Fanny Luor[1], Jill Cao[2], Catherine Law[3], Michael Beswetherick[1], Sheridan Long[2], Margaret Burnett[2], Andrew J. Ko[1]

University of Washington
The Information School[1]
Seattle, Washington, USA

Oregon State University
School of EECS[2] and STEM Academy[3]
Corvallis, Oregon, USA

Summary By-Fardina Fathmiul Alam
SWE 795, Spring 2017

Principles of a Debugging-First Puzzle Game for Computing Education

Motivation

- ❑ Although many systems has designed to help people acquire programming skills independently, but few explicitly teach the subject.
- ❑ These existing systems expect from learners to acquire the necessary skills on their own before they can begin creating their own programs from scratch.

Key Idea

- ❑ Based on all of these observation, This paper proposed a principled approach to teach programming using a debugging game called "**Gidget**" based on a unique set of seven design principles.
- ❑ Gidget is a game designed to teach computer programming concepts through debugging puzzles, which can help novice programmers to learn without requiring an instructor, teach them important program understanding and debugging skills, and lead to create their own programs.
- ❑ This paper has highlighted its seven design principles, design prototypes, and evaluated its use in both a laboratory think aloud study and two summer camps.

Seven Design Principles of Gidget

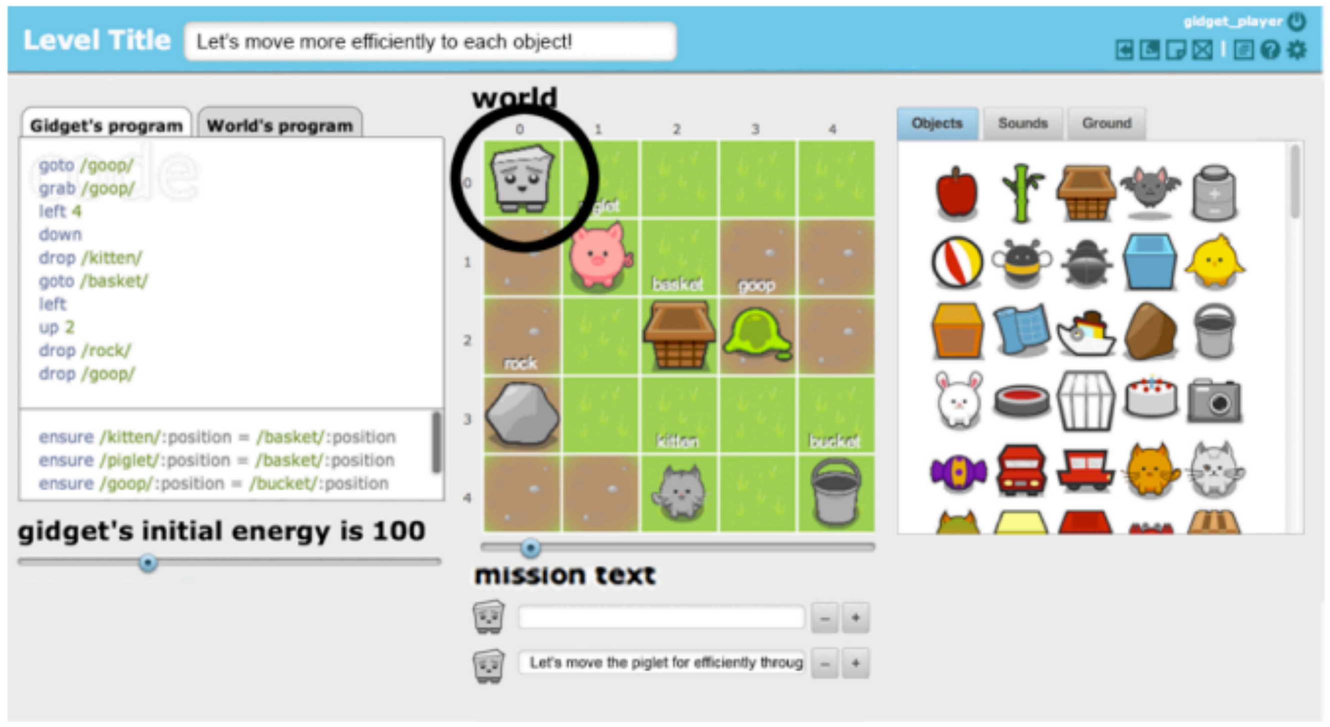
- **P1-debug. Debugging first:** debug existing programs before creating new programs.
- **P2-game. Game-oriented:** The environment should feel like a game to benefit learning.
- **P3-fallible. Computers as helpful but fallible:** Frame computers as helpful but fallible collaborators.
- **P4-goals. Embedded goals:** Focus to one specific game goal – debugging faulty code.
- **P5-instruction. Embedded instructions:** Provide embedded instruction including specific learning objectives, a planned curriculum, set of instructional materials and tasks.
- **P6-help. Scaffolded help:** Deliver, on request, in-game help: problem-solving strategies, higher-level programming concepts.
- **P7-gender. Gender inclusiveness:** Use a gender-neutral protagonist.

Gidget Approach (1/2)

- Gidget the robot was damaged on its way to clean up a chemical spill and save the animals, so it is the players' job to fix Gidget's problematic code to complete all the missions.

- Learners must debug faulty programs to progress through the game, which are set up in modules to teach specific computer programming concepts.

- Once all the levels are complete, learners are given the option to further engage in the game by creating their own levels that can be shared with their friends and family.



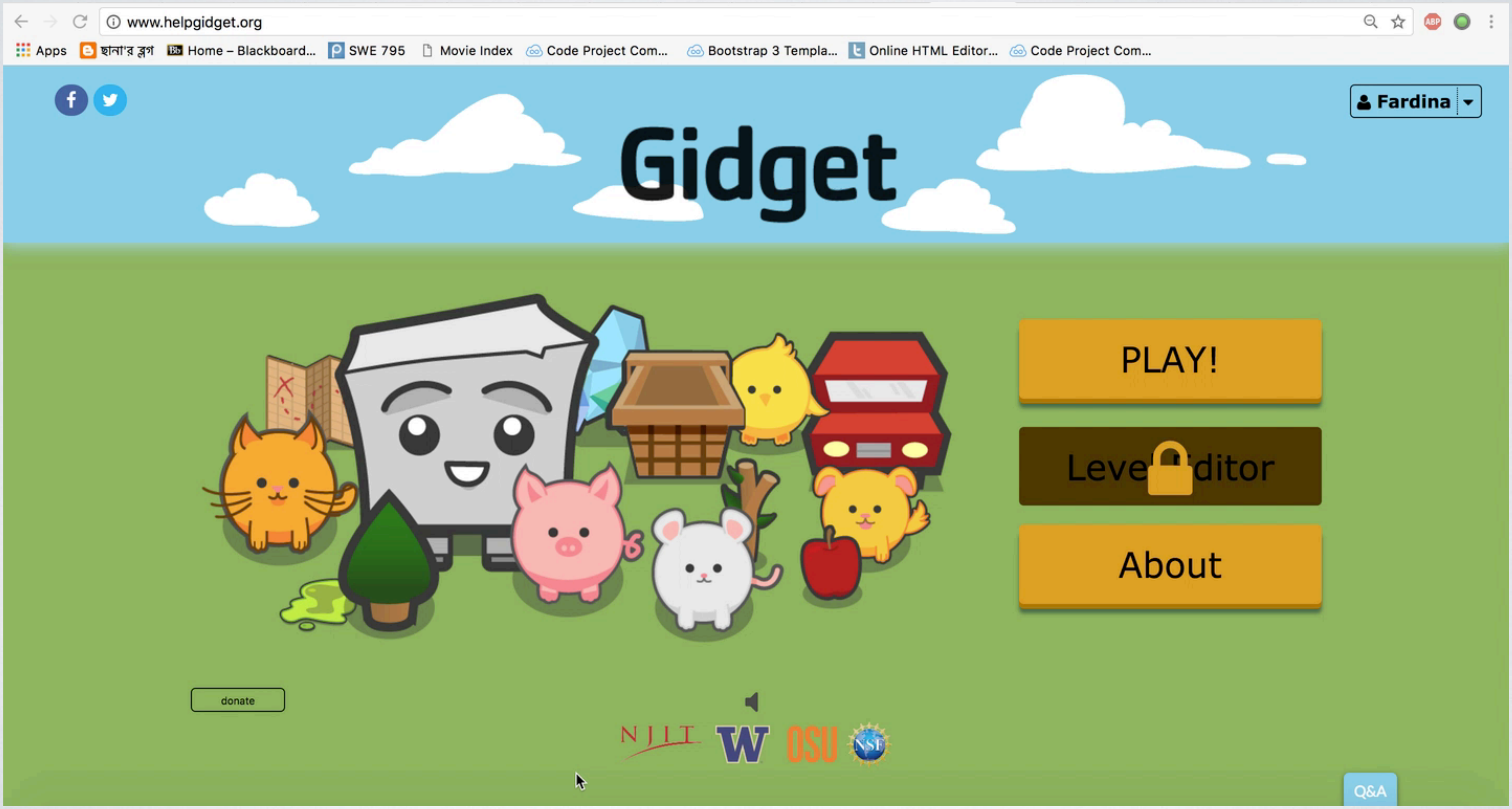
```
goto /goop/
grab /goop/
left 4
down
drop /kitten/
goto /basket/
left
up 2
drop /rock/
drop /goop/
```

```
ensure /kitten/:position = /basket/:position
ensure /piglet/:position = /basket/:position
ensure /goop/:position = /bucket/:position
```

gidget's initial energy is 100

Figure 1. Gidget's level design mode (the Gidget character is circled). In this mode, learners design their own levels for others to solve. Players write code (left) that can include graphics (right), and see animated results (middle), and graphics for the level are on the right.

Gidget Approach (2/2)



Experiment & Results

Conduction of two formative studies:

- **A laboratory think aloud study**: Record in-depth interactions with Gidget. (Think aloud data: The algorithm design barriers, the learning phase barriers)
- **Two summer camps** : observe participants play puzzles and create levels.

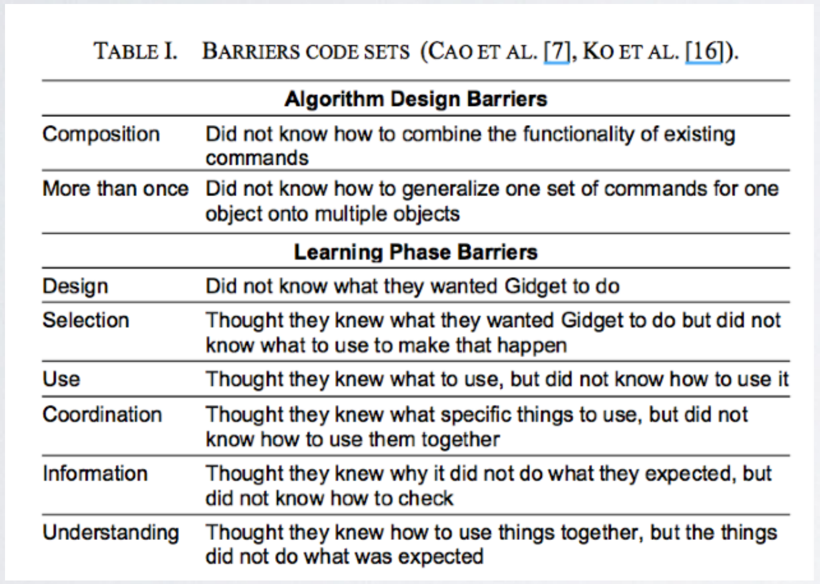


TABLE I. BARRIERS CODE SETS (CAO ET AL. [7], KO ET AL. [16]).

Algorithm Design Barriers	
Composition	Did not know how to combine the functionality of existing commands
More than once	Did not know how to generalize one set of commands for one object onto multiple objects
Learning Phase Barriers	
Design	Did not know what they wanted Gidget to do
Selection	Thought they knew what they wanted Gidget to do but did not know what to use to make that happen
Use	Thought they knew what to use, but did not know how to use it
Coordination	Thought they knew what specific things to use, but did not know how to use them together
Information	Thought they knew why it did not do what they expected, but did not know how to check
Understanding	Thought they knew how to use things together, but the things did not do what was expected

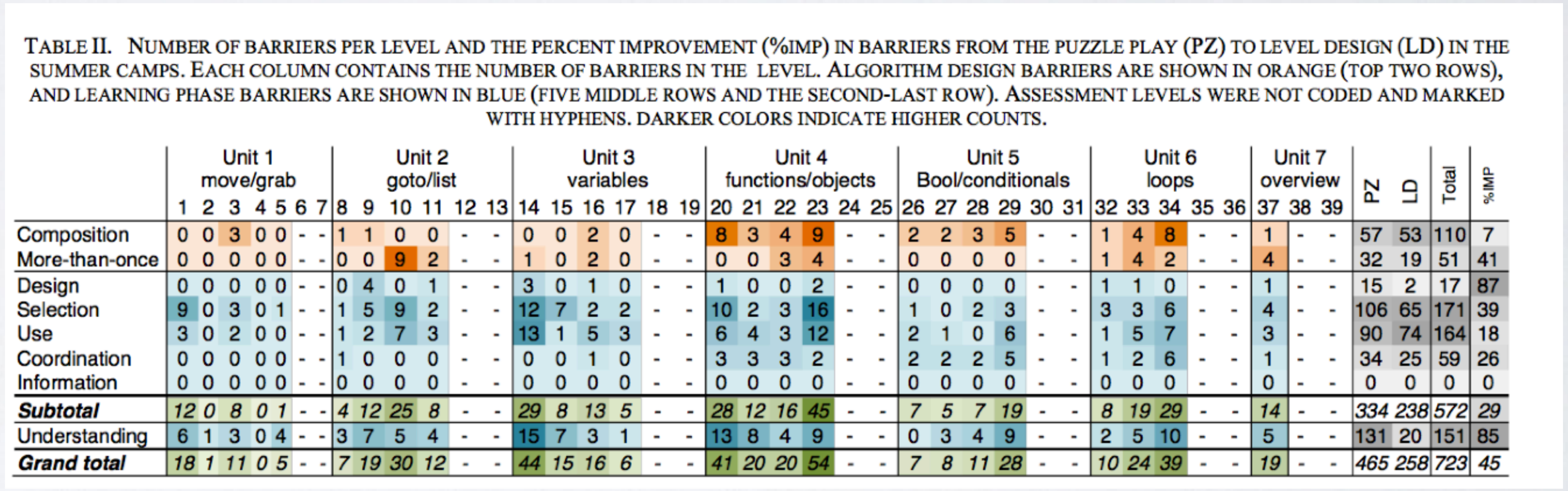


TABLE II. NUMBER OF BARRIERS PER LEVEL AND THE PERCENT IMPROVEMENT (%IMP) IN BARRIERS FROM THE PUZZLE PLAY (PZ) TO LEVEL DESIGN (LD) IN THE SUMMER CAMPS. EACH COLUMN CONTAINS THE NUMBER OF BARRIERS IN THE LEVEL. ALGORITHM DESIGN BARRIERS ARE SHOWN IN ORANGE (TOP TWO ROWS), AND LEARNING PHASE BARRIERS ARE SHOWN IN BLUE (FIVE MIDDLE ROWS AND THE SECOND-LAST ROW). ASSESSMENT LEVELS WERE NOT CODED AND MARKED WITH HYPHENS. DARKER COLORS INDICATE HIGHER COUNTS.

	Unit 1 move/grab							Unit 2 goto/list						Unit 3 variables					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Composition	0	0	3	0	0	-	-	1	1	0	0	-	-	0	0	2	0	-	-
More-than-once	0	0	0	0	0	-	-	0	0	9	2	-	-	1	0	2	0	-	-
Design	0	0	0	0	0	-	-	0	4	0	1	-	-	3	0	1	0	-	-
Selection	9	0	3	0	1	-	-	1	5	9	2	-	-	12	7	2	2	-	-
Use	3	0	2	0	0	-	-	1	2	7	3	-	-	13	1	5	3	-	-
Coordination	0	0	0	0	0	-	-	1	0	0	0	-	-	0	0	1	0	-	-
Information	0	0	0	0	0	-	-	0	0	0	0	-	-	0	0	0	0	-	-
Subtotal	*12*	*0*	*8*	*0*	*1*	-	-	*4*	*12*	*25*	*8*	-	-	*29*	*8*	*13*	*5*	-	-
Understanding	6	1	3	0	4	-	-	3	7	5	4	-	-	15	7	3	1	-	-
Grand total	*18*	*1*	*11*	*0*	*5*	-	-	*7*	*19*	*30*	*12*	-	-	*44*	*15*	*16*	*6*	-	-

	Unit 4 functions/objects						Unit 5 Bool/conditionals						Unit 6 loops					Unit 7 overview			PZ	LD	Total	%IMP
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39				
Composition	8	3	4	9	-	-	2	2	3	5	-	-	1	4	8	-	-	1	-	-	57	53	110	7
More-than-once	0	0	3	4	-	-	0	0	0	0	-	-	1	4	2	-	-	4	-	-	32	19	51	41
Design	1	0	0	2	-	-	0	0	0	0	-	-	1	1	0	-	-	1	-	-	15	2	17	87
Selection	10	2	3	16	-	-	1	0	2	3	-	-	3	3	6	-	-	4	-	-	106	65	171	39
Use	6	4	3	12	-	-	2	1	0	6	-	-	1	5	7	-	-	3	-	-	90	74	164	18
Coordination	3	3	3	2	-	-	2	2	2	5	-	-	1	2	6	-	-	1	-	-	34	25	59	26
Information	0	0	0	0	-	-	0	0	0	0	-	-	0	0	0	-	-	0	-	-	0	0	0	0
Subtotal	*28*	*12*	*16*	*45*	-	-	*7*	*5*	*7*	*19*	-	-	*8*	*19*	*29*	-	-	*14*	-	-	*334*	*238*	*572*	*29*
Understanding	13	8	4	9	-	-	0	3	4	9	-	-	2	5	10	-	-	5	-	-	131	20	151	85
Grand total	*41*	*20*	*20*	*54*	-	-	*7*	*8*	*11*	*28*	-	-	*10*	*24*	*39*	-	-	*19*	-	-	*465*	*258*	*723*	*45*

Figure A. Think-Aloud Study (BARRIERS CODE SETS)

Figure B. Observe participants from Summer Camps

- They calculated each team's percent improvement per barrier type and found improvements in 15 out of 17 camp teams and an overall improvement of 45% from puzzle play to level design.

QUESTIONS FOR DISCUSSION

- Overall reactions
- Would you like to use this technique for computing education for novice learners?
- What limitations does this have?