Fourth Grade Students Reading Block-Based Programs: Predictions, Visual Cues, and Affordances

Hilary Dwyer[1], Charlotte Hill[2], Alexandria Hansen[1], Ashley Iveland[1],
Diana Franklin[2], & Danielle Harlow[1]

[1] Gevirtz Graduate School of Education
University of California
Santa Barbara, CA 93106
{hdwyer, akillian, aockey, dharlow}@education.ucsb.edu

[2] Department of Computer Science
University of California
Santa Barbara, CA 93106
{charlottehill, franklin}@cs.ucsb.edu

ABSTRACT
Visual block-based programming environments allow elementary school students to create their own programs in ways that are more accessible than in textual programming environments. These environments help students write code by removing syntax errors and reducing typing. Students create code by dragging, dropping, and snapping constructs together (e.g. blocks) that are organized by lists, colors, shape, images, etc. However, programming in visual block-based environments is not always simple; in fact, it can become complex quickly. In addition to elements that create code, the visual aspects of these environments provide readers information about what happens, when, and how. Here, we focus on how students used visual cues when reading programs in our block-based programming environment, LaPlaya, a variant of Scratch. Specifically we identified the visual cues students noticed and acted upon. These included not only those that were intended by designers (perceptible affordances), but also those that were not intended by designers (false affordances). Through a detailed content analysis of 13 focus groups with fourth graders we created an initial taxonomy of visual cues in our programming environment and explored how students used these cues to make predictions about provided code, and the types of affordances such cues offered students.

Categories and Subject Descriptors
D.1.7 [**Programming Techniques**]: Visual Programming; K.3.2 [**Computer and Information Science Education**]: Computer Science Education.

General Terms
Design, Human Factors, Languages.

Keywords
Block based languages; graphical programming; computer science education; elementary school; novice programming environments

1. INTRODUCTION
Block-based programming environments have become increasingly popular over the past decade particularly with children and novice programmers. These environments offer colorful, reactive interfaces that reduce the amount of typing needed to write a program, making them well suited for tablet computers and other touch screen devices. Many block-based environments such as Scratch [1] and LaPlaya [2] are centered on characters, or "sprites". In these environments, programs are composed of multiple scripts (segments of code) that are organized by sprite. Not all scripts are visible to the programmer. When programming, the programmer will see four primary areas: a palette of commands (e.g. block lists); the scripting area (e.g. where the programmer connects blocks together); a menu of sprites (e.g. characters available in the program); and a stage (e.g. display visual output). The command palette includes categories of commands organized and displayed into lists by function such as motion or looks. The number of commands available is usually smaller than in textual languages. Next to this area is the scripting area. Scripts are organized by sprite and only those of the selected sprite are displayed.

Many block-based environments are event-driven and implement parallel programming. This means that scripts are programmed to begin when something else happens—a user clicks on a key, or another sprite does something. Often events trigger multiple actions across multiple sprites simultaneously. This means that programmers must keep track of the ways that scripts interact as they work, and how different events trigger action.

Reading and remixing others' programs has increasingly become part of block-based programming environments. Online communities encourage users to view and experiment with others' codes. Scratch [1], Hopscotch [3] and other block-based environments are more than just editors: they also serve as platforms for programmers to easily share their projects online. Scratch's online community encourages "remixing" projects, where users modify another's project. Additionally, programmers may use these online environments to follow coding tutorials, such as "The Hour of Code" on Code.org [4], and modify or expand code snippets provided in the lessons. In the curriculum and interface we designed, students were given sample scripts or partially completed programs to revise or complete [5]. Whether remixing others' code or completing partial code, being able to read block-based programs is as important as being able to create new code.

Reading programs in block-based environments differs considerably both from textual programming environments (e.g., C++ or Python) and other types of reading that children encounter. Block-based programming environments include visual information that may contribute to or, in some cases, hinder comprehension. Children also need to attend to complex and hidden structures that control the flow of the story or game programmed. As a result, we wanted to explore how students understood and read existing code using the visual cues provided by our block-based programming environment (including the scripts and a still picture of the output).

In this paper, we investigated how fourth graders read and made predictions about block-based programs written by other people. Fourth-graders from two schools participating in our computational thinking curriculum—Kids Engaged in Learning Programming and Computer Science (KELP-CS)—made predictions about projects in our programming environment, LaPlaya [2]. We interviewed 13 pairs of students from two local schools about projects conceptually at or slightly above the level they had reached in the curriculum. We asked them to use the information provided to predict and figure out what would happen when a provided program ran.

Here, we present a preliminary taxonomy of the features in LaPlaya that students used to predict what would occur on the stage (the output of a program), functions of individual blocks, functions of individual scripts, and how scripts and sprites were coordinated (understanding of the overall program). Students used the visual features in both expected and unexpected ways. We further analyzed students' explanations through Gaver's construct of technology affordances [6] to identify visual cues that students recognized and acted on, those they failed to notice, and those that they acted on in ways unintended by the developers.

Our analysis demonstrates that, while visual block-based environments reduce syntax errors and typing requirements, they are definitely not simple. As Petre [7] wrote about visual programming more generally:

> Both graphics and text have their uses – and their limitations. Pictorial and graphic media can carry considerable information in what may be a convenient and attractive form, but incorporating graphics into programming notations requires us to understand the precise contribution that graphical representations might make to the job at hand (p. 33).

In block-based programming environments, almost all information and constructs are graphical representations and as yet little research has demonstrated how students use these visuals to aid in program comprehension. In LaPlaya, students utilize a number of intended and unintended visual cues to interpret provided code. We found that, in addition to the scripts and blocks, students used the layout of the stage and characteristics of the sprites to predict program results. Further, students used visual attributes of the programming blocks such as color, shape, and arguments (e.g., words embedded in blocks) to predict how the block or script would function.

2. RELATED WORK

Learning how to program is much like learning how to read: both consist of multiple aspects of literacy such as knowing basic vocabulary, identifying key words, comprehending how series of words construct meaning, and ultimately composing text [8, 9]. Learning to program encompasses the same attributes of learning to read with the added context and structure of the interface and programming language.

Block-based programming environments offer innovative ways to analyze how young students and novices read programs and understand computational concepts (e.g., sequences, events, and operators), practices (e.g., testing, debugging, or reusing), and perspectives (e.g., expressing, connecting, and questioning) [10]. Some work has already demonstrated that students understand programming concepts differently depending on the visual-nature of a particular block-based programming environment. Lewis [11] found that 6th grade students (ages 10-12 years old) appeared to better understand loop construct when using Logo, but the construct of conditionals better in Scratch.

The spatial organization of block-based programming environments makes it easier for programmers to view the output while looking at the scripts. Gross and Kelleher looked at how university students with little to no programming experience understood and reused code in Storytelling Alice [12]. Although participants were not able to fully read and understand the scripts, they were able to match the output to the scripts to find the desired portions of the program. Ferriera et al. [13] found that ninth graders in Brazil using AgentSheets attributed agency to objects (such as an image of a shark) inside of the program, and interpreted the program differently when looking at a program report versus running the program. The program report allowed users to see the whole set of program elements, not just parts associated with a single object/character. Without the visible program report, participants did not understand or notice all the underlying logic and agent in the program. Rader, Brand and Lewis [14] looked at students' understanding of programs in KidSim and found misconceptions stemming from students' views of the objects: children expected objects to behave as they do in the real world and expected the computer to understand the way objects looked (e.g., a picture of a fish) the same way that the children understood the object. Similarly, Pea [15] found that students expected computers to understand programs like people did: remembering code previously executed while running later portions of the program, looking at multiple parts of the program simultaneously, and having implicit knowledge of the program's overall goal.

Though block-based programming environments limit syntax errors and typing requirements, they are complex learning environment and assessing student learning can be challenging [10]. Schulte [9] proposed a Block Model to aid teachers and researchers in identifying different learning paths students may take when comprehending programs "bottom up"—first reading words or text, then making inferences about the relations between blocks, and lastly understanding the overall program structure. This model organized elements of a program by their structure and function. Atoms were language elements and operations of statements. Blocks were regions of interest that syntactically built a unit. Relations referred to connections between blocks. Lastly macrostructure was the overall structure, goal, and purpose of the program. Schultz tested this model with a small sample of potential teachers at one university.

Our work builds on the Schulte's work [9] by examining program and reading comprehension of young students from an ecological perspective. Ecological perspectives [16] assume that objects have actionable qualities that can be acted upon by individuals with the appropriate cognitive and physical resources. For example, a chair has the affordance of being sat in, but only for individuals of the appropriate size. An adult may not be able to sit in a child's high chair, and thus a high chair does not have this affordance for an adult. Affordances are an interaction between the user and the object. In some cases, the actionable qualities are not clear. Norman [17] introduced the term signifier to indicate visual cues to the individual about the affordances of an object. "Push" signs on doors are examples of signifiers, indicating how to access the affordance of opening the door. Gaver [6] applied this work to technology and proposed the idea of perceptible, hidden, and false affordances.

Affordances are, in the context of block-based programming, objects that have possibilities for action. Visual cues provide information about the possible actions. For example, the shape of a block is a visual cue that may indicate which blocks it can connect to or how commands can be linked (affordances). If a child noticed this feature and acted on it, it would be considered a perceptible affordance (from the child's perspective) and an appropriate visual cue. If a child did not notice the feature or did not think that the visual feature had meaning attached to it, it would be considered a hidden affordance. In contrast, students sometimes attach meaning to features that were not intended to impart value such as acting upon visual cues that were not designed to impart information. For example a child might assume (falsely) that the sprite on the far left of the stage would always be the first to act. In reality, the position of sprites is not related to the order in which they appear or act in the program. If this were the case, then position of sprites would constitute a false affordance meaning that students found information that was not actually present.

3. RESEARCH DESIGN

We asked the question, "What perceptible, hidden, and false affordances of a block-based programming environment do students use to read block-based programs?" To answer this question, we conducted focus group interviews in which we presented LaPlaya programs and asked students to predict the outcome.

We interviewed 13 pairs of fourth grade students at two schools, Aguacate and Cabrillo Elementary (pseudonyms), participating in our computational thinking curriculum. Students were interviewed in same-sex pairs: eight pairs of girls and five pairs of boys. Each pair shared a computer and answered questions about and modified three programs written in LaPlaya, the block-based environment used in their computational thinking curriculum.

3.1 Research Context

The fourth graders at both schools, Aguacate and Cabrillo, were participating in our computer science curriculum related to computational thinking and programming. The curriculum includes on-computer and off-computer components: the on-computer exercises are activities in LaPlaya [2] and the off-computer activities relate computational thinking and programming concepts back to every-day life examples.

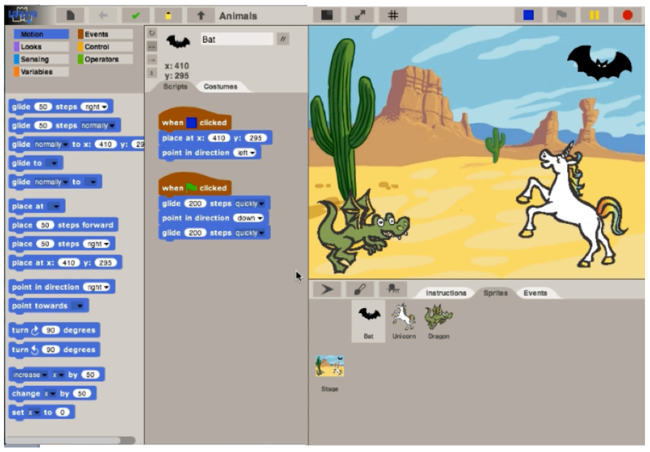
LaPlaya is based on Snap! [18] and Scratch [1]: students use multiple *blocks* of code to create *scripts* that control *sprites*, images of animals or people that students draw or import. In our curriculum, students were given partially completed projects in LaPlaya that already contained some sprites and scripts. LaPlaya allows project designers to hide elements of the programming environment—such as sprites, scripts, or block options—to focus students' attention on specific computer science concepts and reduce the cognitive load required to program. As students moved through the curriculum, more blocks and LaPlaya features were made available to them.

3.2 Participants

We interviewed 26 students in pairs at Aguacate Elementary (n = 16 students) and Cabrillo Elementary (n = 10) over several weeks. We paired students by gender, and together they worked on three programming projects in LaPlaya with an interviewer, a graduate or undergraduate student in computer science or education. The participants and interviewers were familiar with the programming interface. At the time of the interviews, the students had completed 3-4 hours of curricular time with the interface.

3.3 Data Collection

We used two sets of interview protocols each containing three projects: the same introductory project followed by two different projects. For this study, we only analyzed students' actions and discourse in the introductory activity, which was identical for all students. In this activity, the interviewer asked students to predict what the program would do if they ran it. Students explored the program while the interviewer asked follow-up probes to further clarify and expand upon what students were saying.



```
when ■ clicked
place at x: 410 y: 295
point in direction left

when ⚑ clicked
glide 200 steps quickly
point in direction down
glide 200 steps quickly
```

Figure 1. Layout of the first project

Figure 1 shows what students saw when they opened the introductory project. On the stage were three sprites: a bat, unicorn, and dragon. Because the bat sprite was selected at the onset, only the scripts for the bat sprite were initially visible. To see the scripts for the unicorn and dragon, the students needed to click on each sprite individually in the sprite corral (lower right of screen). The interviewer would then ask students to describe what they expected the program to do when it ran. Students were encouraged to interact with the program as long as they did nothing to prompt action on the stage (e.g., click sprites or the green flag).

This activity took approximately 10 minutes to complete, though the complete interview lasted 30 minutes. We captured students interacting with the computer through a combination of video, table microphones and screen recordings of the interface. We combined the video and screen recordings for each pair of students, transcribed the interviews verbatim, and coded the transcripts to find common threads throughout the interviews.

3.4 Data Analysis

We analyzed the focus groups in three iterations examining smaller parts of the transcripts at each juncture. In the first and second rounds (what students were predicting and student tools for making predictions), we analyzed whole transcripts. In the third round (affordances), we analyzed only the introductory activity, and isolated individual instances of visual cues. This impacted our results directly as the activity selected constrained what visual cues could have been discussed and the types of predictions that students made.

3.4.1 What students were predicting

In our first round of analysis, we focused on the ways students responded to the interviewer's probes about what would happen when the program ran. We open-coded the transcripts and created a domain analysis described by Spradley [19]. Using Spradley's specific analytic steps, we identified cover terms and terms related in specific semantic ways to the cover terms. While Spradley's approach to analysis was created to interpret ethnographic studies, we found it appropriate for understanding the programming interface. Like ethnographers, we were trying to describe an emic perspective. That is, our goal was to understand the programming environment from the child's perspective.

In this process, we identified multiple levels of the program that students made predictions about: individual blocks, single scripts, collections of scripts, and the program as a whole. Figure 2 demonstrates our first semantic relationship. We collapsed these terms to align with the functional structure of LaPlaya. This left us with four codes [20] to describe what student made predictions about in LaPlaya: individual blocks, single scripts, multiple scripts, and the output (actions and timing of sprites for program).

X is a predictable attribute of a LaPlaya Program
X may include:

- How sprites will act (actions)
- When sprites will act (timing)
- What a script will do
- What blocks will do
- How programs are run

Figure 2. Semantic Relationship - Attribution

3.4.2 Student tools for making predictions

In our second analytic cycle, we developed semantic relationships [16] for each of the codes developed in our first round of analysis. Each took the form, "X is a tool for predicting Y," where X was an aspect of LaPlaya and Y was one of the predictable attributes listed above. We created a list of terms related to this semantic relationship for each of the four predictable attributes (blocks, single scripts, multiple scripts, and output). As we developed these lists, we realized that many of the tools children used were not necessarily related to the scripts or blocks; rather, the tools related to some visual cue embedded in the interface (e.g., physical characteristics of a sprite).

With this notion of visual cues in mind, we created a summative table relating visual cues to what students were predicting. The visual cues listed in this table were developed both through our group conversations about what was possible, and what students actually talked about in the focus groups with this specific activity. Through multiple rounds of systematic coding [20] and discussion, we reduced, added, and combined visual cues into categories until the group reached consensus.

3.4.3 Affordances

In our third analytic cycle, we systematically coded [20] transcripts again by when students made predictions and what tools they drew on to make those predictions using the construct of affordance [6]. For each prediction, we coded for visual cue and type of affordance (perceptible, false, and hidden). Perceptible affordances occurred when students identified an attribute that developers intended to be useful. False affordances occurred when students interpreted a visual element as useful when that aspect did not in fact contain useful information. Hidden affordances occurred when students overlooked an attribute that developers intended to be useful. Due to methodological limitations, we were not always able to determine hidden affordances as students did not see these attributes. If they did not see a tool, they likely did not discuss it during the focus groups and it would not be captured in the transcripts.

3.4.4 Final analysis

Taken together, our final coding scheme included the taxonomy created in the first two rounds of analysis, and the affordances in round three. Researchers coded transcripts by hand as they watched video and screen recordings for each pair of students. They identified each instance when students made predictions about how a program ran and then identified the visual cue or tool used, what aspects of LaPlaya students were making predictions about (block, single script, multiple scripts, and output) and the type of affordance offered (perceptible, hidden, or false).

Four researchers coded transcripts together until internal consistency was reached with this coding scheme about student predications, tools drawn on on to make predictions, and the intended use of the tool [20]. Then two researchers (one each from computer science education) coded transcripts independently. Pairs resolved discrepancies in coding through discussion and at times with the entire group until consensus was reached. Then, transcripts were uploaded into the qualitative data analysis software Dedoose [21] to aid in the development of findings.

4. RESULTS

Our analyses led to the development of two sets of findings. In the first finding, we describe a taxonomy of visual cues embedded in our visual block-based programming environment, LaPlaya. We developed this taxonomy through several iterations of domain analysis [15], group discussions about theoretically possible visual cues, and visual cues that students discussed during the focus groups. We then outline how this taxonomy linked to what students made predictions about in the focus groups during the introductory activity (block, single scripts, multiple scripts, and output).

In our second finding, we connect the visual cues students discussed with the construct of technology affordances [6]. We provide examples from the focus group transcripts of students acting upon affordances when reading provided code in the activity.

4.1 Taxonomy of Visual Cues in LaPlaya

Using our domain analysis, group discussions, and transcripts we created a taxonomy of visual cues embedded in LaPlaya (see left-hand side of Table 1 and Figure 3 as a reference).

We further sorted these tools into categories based on their function in LaPlaya (blocks, scripts, stage, and interface) and subcategories as necessary. Visual cues in parentheses signaled tools that did not impact how a program ran and thus were categorically false affordances (see following section). For example, in Scratch, whether a programmer places a script in the upper right corner of the scripting area or in the bottom left corner has no bearing on when or how that script is run. Thus interpreting the layout of scripts as providing information is always false. Across the top of Table 2, we listed the aspect of LaPlaya that students could make predictions about (blocks, single scripts, multiple scripts, and output). An "X" signified that students cited a visual cue when making a prediction during the introductory activity. Note that there are two shaded columns with no X's (Single Script and Multiple Scripts). This is because our analysis could not identify these types of predictions but they will be part of future work

Table 1. How Students Used Visual Cues in LaPlaya to Predict Aspects of a Program

			What Students Were Predicting			
Visual Cues in LaPlaya			Block	Single Script	Multiple Scripts	Output
Blocks	Word choice	Prior experience with word	X			X
		Word's everyday meaning				X
	Block layout	Block argument	X			X
		Color				
		Shape				
	Same block, other script		X			X
Scripts	Ordering of blocks within scripts		X			X
	(Layout of scripts)					X
	Other blocks in script					
	Other scripts					X
Stage	(Sprites on the stage)	(Physical characteristics)	X			X
		(Orientation)				X
		(Stage position)				X
	(Background)					
Interface	Sprite corral		X			X
	(Costume tab)					X
	(Costume images)		X			X
	(Instruction tab)					X

Note: Parentheses distinguish visual cues that were categorically false affordances. "X" signifies that students used a visual cue when making predictions during the introductory activity. Both "Single Script" and "Multiple Scripts" columns are shaded because the introductory activity did not provide students with opportunities to predict single or multiple scripts though each emerged in the other focus group activities.

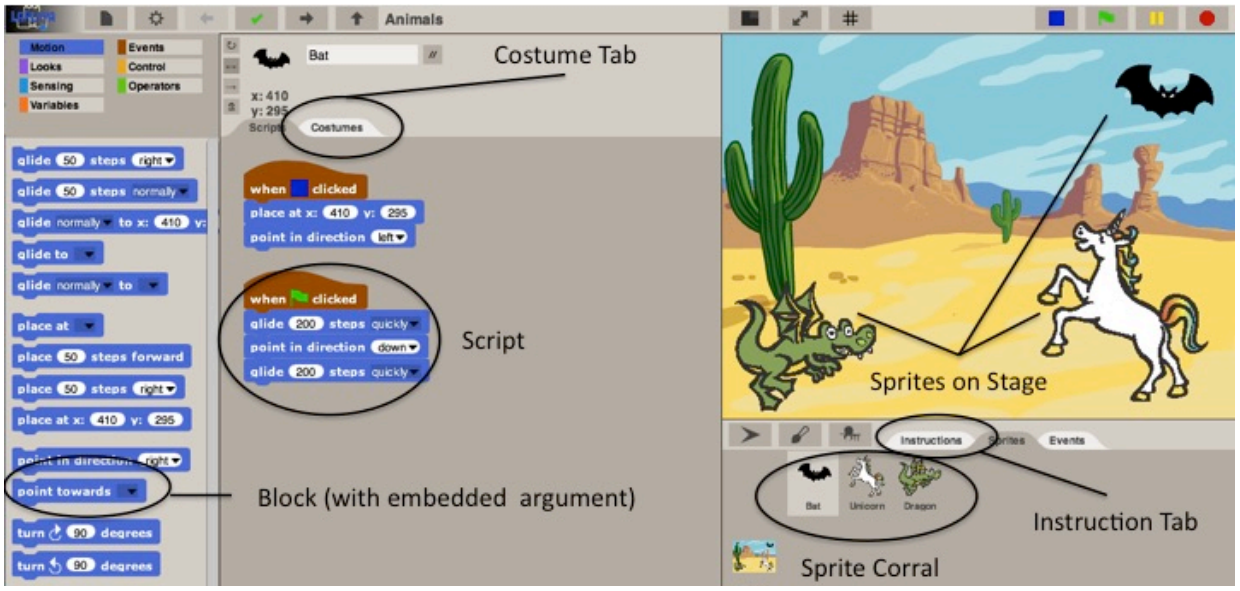
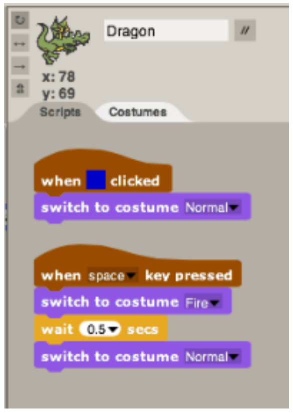


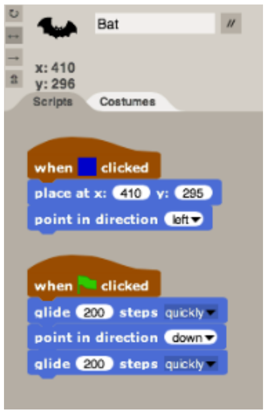
Figure 3. Overview of LaPlaya Interface



```
when [ ] clicked
switch to costume Normal

when space key pressed
switch to costume Fire
wait 0.5 secs
switch to costume Normal
```

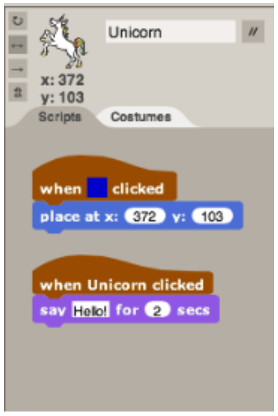
Figure 4. Scripts for Dragon Sprite



```
when [ ] clicked
place at x: 410 y: 295
point in direction left

when [ ] clicked
glide 200 steps quickly
point in direction down
glide 200 steps quickly
```

Figure 5. Scripts for Bat Sprite



```
when [ ] clicked
place at x: 372 y: 103

when Unicorn clicked
say Hello! for 2 secs
```

Figure 6. Scripts for Unicorn Sprite

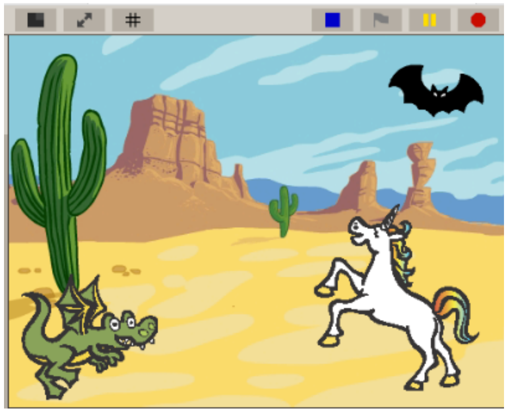


Figure 7. Stage for Introductory Activity

4.1.1 *Categories of Visual Cues*

Block-level visual cues focused on attributes of blocks such as color, shape, and argument. Users could infer block function by the words embedded in a block (e.g. "glide"). Block argument included information that could be passed through the program such as when a user selected from a dropdown menu or wrote in his or her own text (e.g., "say ___ for ___ sec"). Users could look at how a block functioned elsewhere in a project (same block, other script).

Script-level visual cues related to multiple blocks connected together into single or multiple scripts. Users could look at the order of blocks within script or more generally what other blocks were included in a script. They could also get information based on where the scripts were located (layout of scripts) and the way a script functioned in other instances such as under another sprite.

Stage-level visual cues related to the screen in the upper right-hand corner of LaPlaya. Here, users could see sprites (characters) and different backgrounds depending on a project. Users could make predictions based on attributes of the sprites such as physical characteristics (image or icon used for a sprite), orientation (e.g., whether a sprite faced up or down), and stage position (where on the stage a sprite started). Users could also get information from the image displayed in the background. These visual cues were categorically false affordances; attributes of sprite images do not functionally impact a program.

Lastly, interface-level visual cues related to the ways users engaged with the LaPlaya environment not captured in the other categories. The sprite corral was located in the lower right corner and displayed all active sprites by small images or icons. Users clicked on each image to create code for a particular sprite. Within each sprite, users could manipulate the iteration of an image (costumes). Costumes used in sequence create animation on the stage and were listed as icons for each sprite. Users could gain information both from the costume tab listing the costumes, and the particular image for a costume. Finally, LaPlaya included instructions in the lower right-hand side of the interface. These visual cues were categorically false affordances as they did not functionally impact how a LaPlaya program ran.

4.1.2 *Linking Visual Cues to Student Predictions*

The introductory project we selected to analyze constrained what students could make predictions about and what visual cues they discussed. The introductory activity included six scripts total (two for each sprite). One script initialized each sprite and the second script created output on the stage (e.g. action for each sprite). Thus, we could not distinguish when students discussed a single script or output, and we did not provide opportunities for students to make predictions about multiple scripts in the first activity. As a result, these two columns in Table 2, single script and multiple scripts, were blank but still theoretically possible were we to analyze other focus group activities.

In most cases, students used the cues we identified as potential sources of information. For example, when predicting the function of blocks, students discussed the embedded text and arguments in blocks, or how the block was used elsewhere in a program. However, we were also surprised by what visual cues students drew upon to make predictions. When making predictions about blocks, students talked about physical attributes of a sprite, the images of sprites in the sprite corral, and the different costume images provided for a sprite. Though students attributed meaning and acted upon these visual cues, none directly connected to block functionality in LaPlaya.

We found similar patterns when students made predictions about the output of the project – what would happen when the program ran. Students discussed the words and arguments embedded in blocks, and how blocks or scripts were used elsewhere. As we expected, they also read blocks sequentially within a script. However, they attributed meaning to visual cues in unexpected ways. They predicted that some scripts would run first or more quickly depending on where they were located in relation to each other (upper right or left-hand area of scripting area). As well, they used features of particular sprite images to predict the output: physical characteristics, orientation, and position on the stage. They also discussed multiple interface features such as the sprite corral, costume tab, costume images, and instruction tab. In the following section we provide more detailed examples of these different types of visual cues.

4.2 Affordances and Visual Cues

In this section, we provide examples of how students used visual cues to make predictions during the focus groups. We organized these examples by the three types of affordances. Perceptible affordances were visual cues that students recognized and acted on. False affordances were those that students acted on in ways unintended by the developers. Hidden affordances were those they failed to notice. For each type,we provide multiple vignettes demonstrating how students discussed a visual cue and used it to read provided code. All student names are pseudonyms.

4.2.1 Perceptible Affordances

Scripts in LaPlaya are triggered by events. The most common way to run scripts is to click the green flag button, but scripts can also be programmed to run when a user clicks a sprite or presses a key. In the following example, the interviewer asked Kaylee and Ivy to predict what would happen without running the program (clicking the green flag). The students read the embedded argument in visible blocks to predict the output (see Figure 4). This was a perceptible affordance as the students read the blocks and scripts in ways that developers intended.

> Interviewer: What do you think will happen when you click the green flag?
> Kaylee: Ready. Go. [Program will run].
> Ivy: It [the program] will go [run].
> Interviewer: It will go [run] …does anything else make the program run?
> Kaylee: Maybe clicking on the dragon because it says…oh, no! Space key.
> Interviewer: How did you figure out the space key?
> Kaylee: Because it [the block] says "When Space Key Pressed."

Kaylee quickly recognized by reading the provided scripts that the "When Space Key Pressed" block controlled how to run that script. This was an example of students using block arguments to make predictions because "space key" was a dropdown option that also included other keys on the keyboard.

In the next example, Richard and Bryan predicted the output of the program by reading the blocks sequentially for all three sprites. Richard also recognized that he needed to click through the different sprites to see the visible scripts (e.g., sprite corral). Both students used block arguments (e.g. number of steps and direction) and words embedded in blocks (e.g. costume "fire") to make predictions.

> Interviewer: … So before you click on anything, without running the code, what do you think the sprites will do?
> Richard: … with just reading them?
> Interviewer: Yep, just reading the code.
> Richard: Well first of all … can we go through each one [sprite in the sprite corral]?
> Interviewer: Yep! Go through each one [sprite].
> Richard: … when you click the [get] ready [button], they [the sprites] all go back to where they are …When you click the [green] flag, the bat will glide, to right here [points to stage] –
> Bryan: [overlap] 200 steps.
> Richard: Yeah and [move] down, and then [move] over right here. And then the unicorn will say hello and get placed back here like that and say hello. And then the dragon, when you click it, it'll go back to where it is and then it'll switch to fire I'm guessing, and then it'll wait point five, like half a second and then switch back to [].

Richard and Bryan used multiple, perceptible affordances to make predictions about how the program ran. These visual cues (block word choice, block argument, ordering of blocks within scripts, and sprite corral) were intended to be useful when reading LaPlaya programs, and both students acted upon the affordances in expected ways.

4.2.2 False Affordances

Students also acted on visual cues in ways unintended by the developers. These visual cues such as the sprites on the stage or parts of the interface imparted information to students as they made predictions. However, in most cases these visual cues did not functionally impact the program.

In the following, Kaylee and Ivy were predicting what the dragon, unicorn, and bat sprite would do when the program ran (see Figure 4, 5, 6, and 7 for visible scripts and stage setup). Both students looked at the sprites, their attributes, and their location on the stage – instead of the scripts – to predict what they would eventually do when running the program.

> Interviewer: Ok. So, how did you figure out which ones doing what?
> Ivy: …This one's [the bat's] the highest, so I assumed that it must be going down. And, this [the dragon] must be gliding. And this is the only one [the unicorn] that's actually stepping.
> Interviewer: So, they look like they're about to do something?
> Kaylee: Yeah.

As illustrated in the transcript, Ivy predicted that the bat would move downwards because it was positioned in the background's sky (stage location and orientation of sprite on stage). Also, she predicted that the unicorn would take a step because its feet were in the air (physical characteristics of sprite on stage).

In the following, Ethan and Luis also made predictions using attributes of the sprites rather than reading the scripts.

> Interviewer: … So without running the code in this activity, what do you think will happen? What will the sprites do?
> Ethan: I think the bat will start flying to the cactus.
> Luis: The dragon's gonna eat the horse. I think that dragon wants to eat, needs to get to the horse [points to unicorn]

Ethan predicted that the bat sprite would fly across the stage because already located in the upper, right corner of the stage. As well, since the sprite was a bat he concluded that it would "start flying to the cactus" located on the left side of the stage (physical characteristics and stage location of sprite on stage). Luis described how the dragon would move to the unicorn. Since the dragon sprite appeared predatory, he inferred that the dragon would "need to get to the horse" (physical characteristics and stage position of sprite on stage).

The attributes of sprites on the stage do not functionally impact how the program ran, only the associated scripts impart change. However, as these examples demonstrate, students associated the actual image (bat, dragon, or unicorn) with their prior knowledge of each character. They then used this prior knowledge to predict how the program ran. These were all false affordances because students were acting on visual cues not intended to be useful by designers.

4.2.3 Hidden Affordances

Hidden affordances were challenging to identify methodologically with our research design. Hidden affordances existed when a child did not notice a LaPlaya feature or did not think that the visual cue had meaning attached to it – in both cases, evidence of the hidden affordance would be the absence of student talk. As well, we as environment and curriculum developers possessed subjective, insider perspective about what visual tools students should or could be using as they read code.

As a result, there may have been more hidden affordances than we found in this particular analysis. Below we provide one vignette in which students overlooked a LaPlaya feature initially and later in the interview decided that feature was helpful in making predictions. Because they found the tool later, they could articulate that they had not identified it earlier, a rare instance of students describing hidden affordances.

In the following, Kevin did not originally perceive that he could click on sprite icons in the sprite corral to view individual sprites' scripts. He was originally confused about the visible scripts, and how to associate scripts with a particular sprite. He overlooked the affordance of clicking on the sprite icons in the sprite corral. It was not until the moderator asked what the unicorn would do that Kyle eventually realized what he had overlooked. In this event, Kyle vocalized his confusion, allowing us to label this a hidden affordance.

Interviewer: … Ok, what do you think it [the bat] will do?
Kevin: I'm not sure if it [bat] will do anything because, well…I don't know which one [script] will be the bat.
Ben: This one [script] is [for] the bat.
Kevin: No, but like, on there [points to scripts]. Are they all for the bat?
Ben: ....(overlapping) That one [script]. This is the dragon and this is the bat.
Kevin: So I think the bat might not do anything. [Kevin does not associate scripts for the bat sprite]
Kevin: Maybe.
Ben: Yeah, maybe we point the dragon to the bat and the bat is going to … go away.
Interviewer: … so what do you guys think the unicorn is going to do?
Kevin: Point left and then, maybe just, might not do anything because it's already pointing left.
Ben: Left is on the (??) and right is to the – right here. And this is right.
Interviewer: So for the sprites, it's easier- this is one is for the (bat) right?
Kevin: Oh! It's all for the bat. [Kevin recognizes all scripts refer to bat]
Interviewer: Yes, so how do you see what the unicorn will do?
Kevin: Oh, you have to click on the unicorn.
Interviewer: So what do you think will happen when you click on the unicorn?
Kevin: It will say hello.

In this example, the students initially overlooked that they could click through sprites in the sprite corral. Kevin did not use this feature as designers had intended, a false affordance. As the interview progressed however he realized that he could click through the different icons and change what scripts were visible.

5. DISCUSSION

Students in our focus groups recognized that blocks and scripts were important tools in predicting what would happen in a visual block-based program. They read provided scripts to inform their predictions; and nearly always, they recognized that the scripts held information that would aid in figuring out what would happen in the program. However, students did not use scripts as the only tool or even as their first tool. Students attempted to use information on the stage (e.g., placement of sprites), to imagine what the characters could do based on sprite characteristics (e.g., a bat image flies horizontally), and visual information related to blocks (e.g., whether it contained a dropdown menu). In some cases this information was effective and provided information that was usable. In other cases, the visual cues were distracting.

We analyzed how students read code in our block-based programming environment – LaPlaya – that had been adapted from Scratch for our particular student population and learning objectives. The taxonomy we created reflects the particular visual cues and affordances embedded in our interface. However, since LaPlaya was a variant of Scratch, the taxonomy may reflect tools in comparable Scratch-based environments. Many of the attributes implemented in LaPlaya (e.g., blocks, sprites, scripts, and stage) are available in other environments.

Though our specific findings are LaPlaya or Scratch specific, they offer two major contributions more generally to the computer science research community. First, reading code in visual block-based programming environments is complex. There are many components to a single program that could be analyzed (e.g. single blocks, single scripts, single scripts with multiple sprites, multiple scripts across multiple sprites, etc.). Young students like those in this study may be better able to discern, interpret, and read individual aspects of a visual blocked-based program but not all. Young students reading these programs may need explicit instruction about how the different attributes work independently and together alongside the development of their own programs.

Second, students use the visual nature of block-based programming environments in both intended and unintended ways. In many of these environments, users create code (e.g., scripts) alongside the output (e.g., stage) within a single interface. As designers and developers add features to the interface in the hopes of facilitating the creation of code (e.g., such as listing sprites by images) they are also creating sources of (mis)information for users. In some cases young students may overlook provided features (hidden affordances); in other cases, they act upon features that do not have actual use in the environment (false affordances). Curriculum developers and researchers should consider all visual cues and affordances – hidden, false, and perceptible – when analyzing how novices and young students read projects in these environments.

Our goal was to understand the information on the screen that a typical student might interpret as useful to understand the program. Here, our unit of analysis was the interface, not individual children. So while we described vignettes from individual focus groups to elaborate on our findings, the findings were not used to describe how well an individual or pair of students understood the LaPlaya programming environment. We sought to illuminate the complexities within a block-based programming environment for young students reading projects. To simplify the analysis, we also focused on a single project in LaPlaya that constrained the visual tools students could discuss during the interviews. Next steps in this area could focus on using a variety of projects that align with particular aspects of a taxonomy like the one described we created. As well, future work could focus on which visual cues are used most commonly used by different groups of students such as by age, background in programming, and gender.

6. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation CE21 Award CNS-1240985. We are grateful to the teachers and children who participated in this project.

7. REFERENCES

[1] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: Programming for all. *Commun ACM*, 52, 11, 60-67.

[2] Hill, C., Dwyer, H. A., Martinez, T., Harlow, D., & Franklin, D. (2015). Floors and flexibility: Designing a programming environment for 4th – 6th grade classrooms. In *SIGCSE '15*. Kansas City, MO: ACM.

[3] Hopscotch Technologies. (2015). Hopscotch – Programming made easy! Make games, stories, animations and more! (Version 2.12) [Mobile application software]. Retrieved from https://www.gethopscotch.com

[4] Alvarado, C. (2014). CS Ed Week 2013: The hour of code. *ACM SIGCSE Bull*, *46*(1), 2-4.

[5] Franklin, D., Harlow, D., Dwyer, H., Henken, J., Hill, C., Iveland, A., Killian, A., & Development Staff. (2014). *Kids enjoying learning programming and computer science (KELP-CS)- Module 1 Digital Storytelling.* Available at https://discover.cs.ucsb.edu/kelpcs/educators.html

[6] Gaver, W. W. (1992). Technology affordances. In *CHI '91*. New York, NY: ACM.

[7] Petre, M. (1995). Why looking isn't always seeing: Readership skills and graphical programming. *Commun ACM*, *38*(6), 33-44.

[8] Pea, R., D. & Kurland, D., M. (1984). On the cognitive effects of learning computer programming. *New Ideas Psychol, 2* (2), pp. 137-168.

[9] Schulte, C. (2008). Block model – an eductional model of program comprehension as a tool for a scholarly approach to teaching. In *ICER '08*. Sydney, Australia: ACM.

[10] Brennan, K., & Resnick, K. (2012). *New frameworks for studying and assessing the development of computational thinking*. Presented at AERA '12. Vancouver, BC.

[11] Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. In *SIGCSE '10*. Milwaukee, WI: ACM.

[12] Gross, P., & Kelleher, C. (2009). Non-programmers identifying functionality in unfamiliar code: Strategies and Barriers. *J Visual Lang Comp, 21*(5), 263-276.

[13] Ferreira , J. J., de Souza, C. S., de Castro Salgado, L. C., Slaviero, C., Leitão , C. F., de F. Moreira, F. (2012). Combining cognitive, semiotic and discourse analysis to explore the power of notations in visual programming. In *VL/HCC '12*. Innsbruck, Austria: IEEE.

[14] Rader, C., Brand, C. & Lewis, C. (1997). Degrees of comprehension: Children's understanding visual programming environment. In *CHI '97*. Los Angeles, CA: ACM.

[15] Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *J Educ Comput Res*, *2*(1), 25-36.

[16] Gibson, J. J. (1979). *The ecological approach to visual perception.* New York, NY: Houghton Mifflin.

[17] Norman, D. (2002). *The design of everyday things*. New York, NY: Basic Books.

[18] Garcia, D., Segars, L. & Paley, J. (2012). Snap! (build your own blocks): tutorial presentation. *J Comput Sci Coll 27*(4), pp. 120-121.

[19] Spradley (1979). *The ethnographic interview.* Forth Worth, TX: Hancourt Brace.

[20] Saldana, J. (2013). *The coding manual for qualiative researchers* (2nd ed.). Thousand Oaks, CA: SAGE Publications, Inc.

[21] Dedoose. (2014). Web application for managing, analyzing, and presenting qualitative and mixed method data (Version 5.0.11). Los Angeles, CA: SocioCultural Research Consultants, LLC. Retrieved from www.dedoose.com