Investigating the Role of Purposeful Goals on Novices' Engagement in a Programming Game

Michael J. Lee and Amy J. Ko
The Information School | DUB Group
University of Washington
Seattle, Washington, USA
{mjslee, ajko}@uw.edu

***Abstract* — Engagement is a necessary condition for learning, especially for challenging topics such as computer programming. Previous studies have shown that changes to the presentation of educational game elements can significantly affect learners' engagement to continue playing. We hypothesized that changing the presentation of the data referred to in a game's goals would influence the purposefulness of the goals, thereby affecting players' motivation to achieve them. To test this, we designed a game called Gidget, where the eponymous robot protagonist needs assistance to correct its code to complete its missions. In a three condition controlled experiment, we manipulated the presentation of the game's data elements and goals, and assessed the impact on several measures of player engagement. We tested our game with 121 self-described novice programmers recruited on Amazon's Mechanical Turk and found that, when given the option to quit at any time, those in the condition using vertebrate data elements completed twice as many levels as those using abstract data elements. Moreover, the participants using the vertebrate and invertebrate elements spent significantly more time than those using the abstract data elements playing the game overall. Finally, although players across all three conditions spent similar amounts of time interacting with the game's user interface, editing code, and completing and attempting levels, we found that those in the abstract condition were much more likely to quit, especially on difficult levels. These findings suggest that the presentation of game elements can strongly affect the purposefulness of goals, which may play a significant role in keeping self-guided learners engaged in learning tasks.**

Keywords – Programming, education, engagement, games

I. Introduction

Engagement is a necessary condition for learning [12], especially for challenging topics such as computer programming [6]. One promising way to both engage learners and ensure learning are educational games [12,31]. For example, in our prior work, we have explored a debugging game called Gidget, in which the player works with a damaged robot to diagnose and correct its faulty programs, helping it complete its mission objectives [22]. Games such as these can provide immediate, interactive feedback that builds learner confidence, allowing them to assess themselves and identify skills they need to develop [32]. Games like Gidget are similar to the collaboration between learner and tutor, where the tutor (i.e. the game), provides personalized feedback to help the learner complete their objective(s). As with a human tutor, effective games can be engaging by providing players with clear objectives and the skills needed to achieve them [12].

TABLE I. Conditions, Goals, and Images of the First Level

Condition	Goal (Level 1)	Respective Game Images
Abstract	*block on bin*	
Invertebrate	*beetle on jar*	
Vertebrate	*kitten on basket*	

Such engagement may only occur, however, when objectives are meaningful (i.e. have a purpose) to the learner. Although these effects have been examined in formal educational settings [21,25], much less is known about their effects in informal contexts, especially in the space of educational games. For example, in studies of attrition in CS1 courses, one of the most common factors behind dropouts were that students did not feel that the programs they were writing solved meaningful problems [25]. This trend was verified by Layman et al., who argue for more meaningful assignments after finding that 41% of 200 surveyed CS1 course projects had no practical context (e.g. sorting a list of meaningless numbers) [21]. In our prior work using Gidget, we investigated this effect in the context of games, attempting to manipulate the extent to which the player believed they were helping Gidget, comparing a robot with a face that used personal pronouns such as "I" and "we" to one that had no face and used conventional error messages. We found that players who worked with the personified robot were significantly more likely to report wanting to help it and completed twice as many levels in a similar amount of time as the other group [22].

Whereas in our previous study we investigated the effect of the visual presentation of the program interpreter, in the present study we investigate the effect of game goals, manipulated by the presentation of data elements. Recent work has demonstrated that humans have evolved to empathize with animals [4], suggesting that players may attribute more purpose in the goals working with animate data objects, particularly vertebrates [2]. In Gidget programs, data are the objects that the robot scans, analyzes, and moves, such as those in Table 1, and these objects are directly tied to the goals that the player is trying to accomplish. Goals in the game include transferring spilled chemicals into containers, checking attributes of objects, and moving animals to safety. We hypothesized that changing the presentation of the data referred to in these goals

would influence the purposefulness of goals, thereby affecting players' motivation to achieve them, especially as goals become increasingly difficult to accomplish.

Our experiment, which involved 121 rank novice programmers, asked participants to play the game (Fig. 1) until they wished to quit, enabling us to measure engagement as play time and the number of levels completed. To manipulate the purposefulness of the goals, we designed three versions of the game involving the three different kinds of objects shown in Table 1: abstract, invertebrate, and vertebrate. We found that those in the condition interacting with vertebrate data elements completed significantly more levels than those in the other two groups in a similar amount of time. In addition, although we found that the participants using vertebrates played the game significantly more overall, there were no significant differences in play time on individual, attempted levels across conditions.

II. Related Work

Educators use engagement to improve learning. According to engagement theory, engaged students learn at high levels, better grasp what they learn, and retain that knowledge [18]. Experts agree that increasing student engagement in educational topics is key to success [8,14]. Since engagement in learning activities is connected with tasks perceived to be meaningful [18], it is closely related to motivation.

There are several studies demonstrating that working towards *meaningful goals* positively affects engagement in gaming and learning contexts. Bowman encourages learners to play an active role in their engagement by having them pursue goals they find personally meaningful [3]. Similarly, Malone argues that a key element for creating enjoyable educational games is to provide clear goals that students find meaningful [23]. Vroom reported that the valence of an activity (or the attractiveness of outcomes) plays a major role in effort expenditure [37]. Meaningful goals have been found to be particularly important for women [25], minorities [11], and millennials (those born after 1982, including men) [33]. However, Layman et al. find that current CS1 courses lack meaningful projects [21]. Our work extends these studies done in formal learning settings, exploring how the representation of particular game elements affects engagement in achieving game goals in an informal learning context.

Other works support the constructionist approach to learning, supporting children's engagement by having them work on relatable and personally meaningful projects [34]. Kelleher et al. [19] were one of the first to demonstrate that opportunities and affordances for storytelling can significantly improve learners' motivation to program by making projects more personally relevant. Several systems offer varying instantiations of objects to reinforce the motivating nature of storytelling. For example, MOOSE Crossing invites learners to create characters and spaces in a virtual, multi-user text-based world [5]; more recently, work by Ryokai et al. had children learn programming by partnering with a physical robot that acts out stories through personal drawings [36]. Storytelling Alice [19] and Scratch [24] also enabled learners to tell and share stories. Our work follows these traditions, but provides learners with the story, allowing them to contribute to its progress.

Concrete representations may provide context for making goals more purposeful. For example, an abstract task such as,

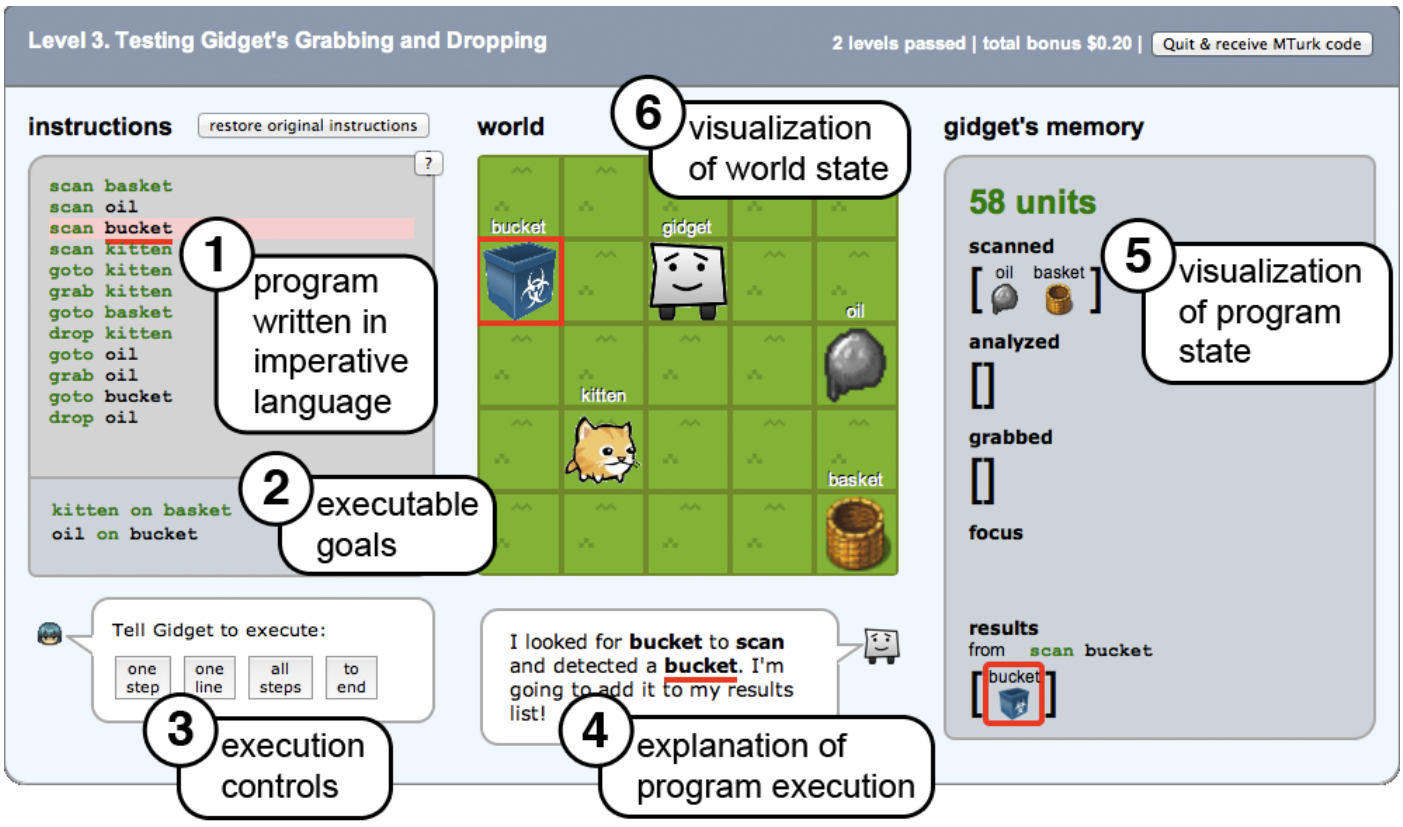

Figure 1. Gidget (showing the *vertebrate* condition), in which players help a damaged robot fix its broken programs. The goal of this level is to help Gidget fix the broken program to clean and secure the area by moving the *oil* to the *bucket,* and the *kitten* to the *basket*.

"sort this list of integers" [21] does not convey the same meaning as "triage these injured patients." This strategy of using concrete representations to explain abstract concepts is a common pedagogical practice and supported by large organizations such as the National Council of Teachers of Mathematics (in this case, using concrete representations to explain abstract concepts) [16]. However, evaluation of the effectiveness of this practice on learning is scarce and largely anecdotal [16], leading to many conflicting viewpoints within the education community. For example, Ainsworth et al. [1] found that elementary students who were shown concrete visual aids outperformed peers who were not shown the visual aids in a math activity. Conversely, other studies in mathematics education found concrete instantiations introduce irrelevant distractors that hinder students' ability to apply their knowledge to similar problems [17]. Still others support and demonstrate the effectiveness of a combined approach such as concreteness "fading," where students are first shown a concrete example and subsequent examples become more abstract [13]. All these strategies, which affect learning, may have direct implications on student engagement [27]. Our work extends these studies done in formal learning settings, exploring how concrete or abstract instantiations of data elements affect engagement in achieving game goals in an informal learning context.

III. Method

The aim of our study was to investigate how the purposefulness of goals, manipulated by the visual representation of data elements and their labeling, affects learners' voluntary engagement. To do this, we designed Gidget, shown in Fig. 1. The game asked learners to help a damaged robot fix its faulty programs in order to accomplish its missions. Our study had three conditions: the control condition involved data elements that were abstract and inanimate (Table 1). The two experimental conditions involved data elements that were concrete and animate; in one condition these were vertebrates and in the other they were invertebrates (Table 1). We hypothesized there would be a difference in levels completed and time spent on levels in the three conditions, with a preference for working with vertebrates. The study was a between-subjects design with 41 participants in the vertebrate condition, and 40 each in the invertebrate and abstract conditions. Participants were recruited on Amazon.com's Mechanical Turk and offered $0.40 for completing at least one level, and an additional $0.10 for every level completed onwards. The number of levels completed was displayed in the upper right corner of the interface, along with a button giving the participants the option to quit at any time (Fig. 1). The key dependent variable in our study was engagement, which we operationalized as the number of levels completed, the time spent on each level, and the use of different UI elements such as the code editor. In this section, we describe the game, the experimental manipulations, our dependent variables, and our experimental procedure.

A. The Game

Our game, called Gidget (shown in Fig. 1), is a web-based, HTML5 application. Learners are guided through a sequence of levels that teach the design and analysis of basic algorithms in a simple imperative language designed specifically for the game. A simple story motivates the game: a small robot capable of identifying and solving problems with programs has been deployed to clean up the area and shut down a factory that has gone awry. Unfortunately, the robot was damaged during transportation, and now struggles to complete its missions, generating programs that almost accomplish its missions, but not quite. It is up to the learner to help the robot by figuring out and fixing its problematic code. In this sense, the learner and the robot are a team, working together to complete levels and ultimately shut down the hazardous factory.

The primary activity in the game is to learn how to communicate with the robot via commands to help it accomplish a series of goals. The levels, goals, language, and user interface (UI), however, were designed to teach specific aspects of algorithm design. The first 9 levels focus on teaching the 7 basic commands in the robot's syntax grammar (Table 2) as well as variations on how these commands can be written. These levels each contain some syntax error that learners must understand and correct by inspecting the program, executing it, and optionally reading Gidget's explanations of his actions at each step in the code (e.g. Fig. 1.4). The subsequent 9 levels teach design patterns for composing these commands to achieve more powerful behaviors, each containing some semantic error. Each level includes one or more goals (Fig. 1.2), which are executable expressions that must all be true after program execution to proceed to the next level. Each goal is on a single line predicate, with corresponding references to the data elements in the world (e.g., Fig. 1.2 and 1.6: **kitten** and **bucket**).

Table 2 explains the robot's commands. Learners had access to a similar syntax reference, but with simpler explanations, through the *?* button at the top right of the editor (Fig. 1.1). Each of the 7 commands could be followed by a *','* and subsequent command, allowing the robot to iterate over a set of things with a given name. For example, if there were multiple kittens in Fig. 1, the command **goto kitten, grab it** would iteratively go to each kitten, grab the kitten, and then go to the next kitten. The current object in a set is pushed onto the focus stack (Fig. 1.5); it always resolves to the object at the top of this stack. The results stack tracks matching names for each

TABLE II. Gidget command Syntax and Semantics

scan *thing*
Enables Gidget to **goto** all things with name *thing*. Scanned things are added to the set named *scanned* in Gidget's memory.

goto *thing1* [avoid *thing2*]
Moves Gidget to all of the things matching the name *thing1*, one square at a time If a thing to **avoid** is given, for each step that Gidget takes, he attempts to find a path that stays at least 1 square away from things with the name *thing2*.

analyze *thing*
Enables Gidget to **ask** all things with name *thing* to perform an action. Analyzed things are added to the set named *analyzed* in Gidget's memory.

ask *thing* to *action thing* *
Causes *thing* to perform *action,* if action is defined. Zero or more things are passed as arguments. Gidget's execution is suspended until the thing asked has completed requested action.

grab *thing*
Adds all things with name *thing* to the set named *grabbed* in Gidget's memory, removing them from the grid and constraining their location to Gidget's location.

drop *thing*
Removes all things with name *thing* in that were previously grabbed from the set *grabbed* set.

if *thing* is[n't] *aspect*, *command*
For each thing with name *thing* that has been analyzed, execute the specified command if that thing contains an aspect of name *aspect*.

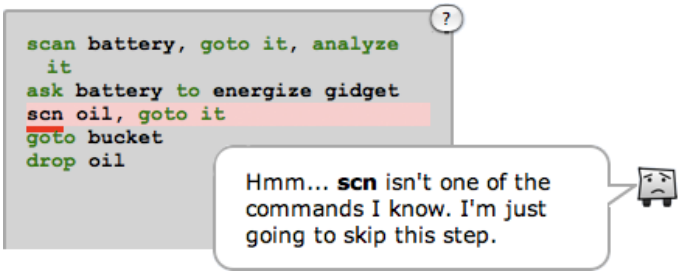

Figure 2. Code (left) and Gidget's response (right) describing the execution step using personal pronouns to increase agency.

command's *thing* query. Each of these data structure views are updated after each step in the program, providing learners with a visualization of Gidget's state.

In the game, Gidget programs are primarily capable of findings things in the world (Fig. 1.6), going to them, checking their properties, and moving them. In some cases, objects have their own abilities, which can be invoked like a function. After each execution step, the effect of these commands is shown in the memory pane (Fig. 1.5) and explained by the robot (Fig. 1.4), teaching the semantics of each command to players. Each step costs 1 unit of energy (above Fig. 1.5), requiring learners' to carefully consider how to write their code to complete each level within the allotted number of energy units.

The robot is detailed in its interpretation of each command, explaining what action it has taken after each step (Fig. 1.4) and visualizing changes to the data structures it maintains in memory (Fig. 1.5). When it arrives at an unrecognized or incomplete command, it explicitly highlights the missing information and explains what interpretation it is going to make before proceeding (e.g. Fig. 2). Upon execution of a line with parsing errors, the system also opens up a syntax guide available from the *?* in the editor (Fig. 1.1), highlighting the rule that the robot guessed was being used. Finally, the robot was given facial expressions (neutral, happy, and sad) shown upon error states and goal completions. It referred to itself and the player using personal pronouns in its feedback such as *"I don't know what this is..."* and *"I never could have done it without you!"* This was based on research showing that personified feedback by an interpreter with agency significantly increased learners' engagement [22].

To aid the players with debugging, the game includes four execution controls for the code: *one step*, *one line*, *all steps*, and *to end* (Fig. 1.3). The *one step* button evaluates one compiled instruction in the code, as a breakpoint debugger does, but also displays text describing the execution of the step (Fig. 2). The *one line* button evaluates all steps contained on one line of the code, jumping to the final output of that line immediately. The *all steps* button evaluates the entire program and the goals in one button press, animating each step. The *to end* button does the same as *all steps*, but only shows the program end state, without animating intermediate steps.

The game was compatible on MacOS X, Windows 7, and Ubuntu Linux 10, using Apple Safari 5, Mozilla Firefox 5, and Google Chrome 10. (We were unable to support Microsoft Internet Explorer 9 because it lacked the *contentEditable* attribute, which was used to implement the editor). This may have affected our participant recruiting, since at the time of the study, nearly half of worldwide web traffic was from lesser versions of Internet Explorer [30].

B. *The Three Level Conditions*

The independent variables we manipulated in our experiment were the labels and visual appearance (Table 3) of the objects referred to in the level goals (Fig. 1.2). In the abstract condition, the data elements the player and robot interacted with were designed to be inanimate, abstract objects. This condition was intended to diminish the purposefulness of the goals, separating them from the context of the story. The items in the abstract condition were all blocks of various colors with abstract signs, such as the arrow in Table 3. In contrast, both of the experimental conditions' data elements were designed to be specific, animate, concrete objects. In the vertebrate condition, the data elements included cats, birds, dogs, kittens, puppies, piglets, and rats. In the invertebrate condition, the data elements included beetles, flies, ladybugs, bees, termites, butterflies, and spiders. These conditions were intended to increase the purposefulness of the goals, tying them to the context of the story. Our hypothesis, based on prior work showing that humans empathize and attribute more positive attitudes towards vertebrates [2,4], was that players would ascribe more purpose in saving vertebrates (and perhaps invertebrates) than abstract objects, and therefore complete more levels.

Many of the level goals required Gidget to move certain data elements to the position of another, usually a container like those in the first column of Table 3. The visual representation and names of these data elements were also different in each condition, reflecting the overall theme of its group's other data elements. There were two distinct containers in each condition, including bin/pod, basket/bucket, and jar/bucket for the abstract, vertebrate, and invertebrate conditions, respectively.

C. *Recruitment*

The population we focused on were rank novice programmers, defined as individuals who self-reported that they had never written a computer program. To recruit these individuals, we used Amazon.com's Mechanical Turk (MTurk), an online marketplace where individuals can receive micro-payments for doing small tasks called Human Intelligence Tasks (HITs). Since workers are sampled from all over the globe, MTurk studies can generalize to more varied populations than samples from limited geographic diversity [20]. However, due to the nature of the low monetary compensation and anonymity of the workers, careful consideration has to be taken to ensure the quality of workers' submissions [10,20]. To address this, we required that participants complete at least one level to receive credit for the HIT, ensuring that they actually interacted with Gidget, the code, goals, and the data elements before quitting.

TABLE III. A SELECTION OF DATA ELEMENTS FROM THE 3 CONDITIONS

(inanimate) **abstract**	bin	block	block	tile
(animate & concrete) **invertebrate**	jar	beetle	fly	ladybug
(animate & concrete) **vertebrate**	basket	kitten	bird	dog

D. Pricing and Validation

Our pricing model and validation method was carried over from a previous study [22]. Since our game had a total of 18 levels, we decided to compensate our participants with a base rate and a nominal bonus payment for each level they completed. Participants were instructed that only the first level was required to receive compensation, and that subsequent levels were completely voluntary. Previous studies have found that higher payments do not necessarily equate to better results [15], so we calibrated our payments to established market prices. To do this, we observed MTurk HITs tagged "game" for 14 days. We filtered these HITs to include only those that had an actual gameplay element as the main component (as opposed to tasks such as writing reviews for games). From these HITs, we constructed a list of 'reward' and 'time allotted' values, along with any explicit bonus payments mentioned. Our goal was to set a base reward that was high enough to attract participants, but also as low as possible to minimize participants' sense of obligation to spend time on our HIT. Likewise, we wanted our bonus payments to have a minimal effect on a worker's decision to continue playing the game.

Based on our data, we determined our optimal base reward as $0.30 for starting the HIT, and an additional $0.10 for each level completed. Because we required that players complete at least one level to get paid, the minimum compensation was $0.40. Participants were not informed of the total number of levels to eliminate that factor from their decision to continue playing the game. In addition, we set the ceiling for submission time to 3 hours so that potential participants could gauge the difficulty of the HIT compared to other HITs. Finally, we deliberately avoided mentioning programming in the HIT description to prevent people from self-selecting out because of its association with programming. However, since the HIT description included the words "game" and "robot," we may have introduced some selection bias related to these topics.

To further validate our pricing model and detect defects and usability problems in the game, we conducted a pilot test on MTurk with 29 participants. The pilot study results verified that participants were willing to complete levels and that the system functioned as expected. Based on the information we received, we fixed a few minor data element name inconsistencies.

E. The Participants

Because we deliberately chose not to mention programming in our HIT description, we could not exclude those with prior programming experience from participating. Therefore, we recruited a large sample of 251 participants from MTurk. Of these, 121 met our criteria for being novice programmers, which included all participants who responded "never" to all of the following statements: 1) "taken a programming course," 2) "written a computer program," and 3) "contributed code towards the development of a computer program."

Participants were distributed proportionally among our three conditions by demographics, with no statistically significant differences in gender ($\chi^2(2,N=121)= 1.1$,n.s.), age ($\chi^2(2,N=121)= 3.6$,n.s.), level of education ($\chi^2(14,N=121)= 4.0$,n.s.), or country of residence ($\chi^2(32,N=121)= 30.7$,n.s.). The median age was 26, ranging from 18 to 66 years old. Though we expected there to be a gender sampling bias because the HIT was labeled as a game, our sample included 63 females and 58 males, which is consistent with other MTurk findings that females are major contributors on the site [35]. Participants were primarily from the US (61.6%), and India (14%). The remaining were from the UK (5.6%), Canada (3.3%), and 13 other countries (16%). Consistent with studies of mTurk demographics [10,20], our sample was well-educated, reporting that their highest level of education achieved was: less than high school (1.7%), high school (17.4%), some college (21.5%), an associates degree (8.3%), a bachelor's degree (38%), a masters degree (8.36%), a professional degree (1.7%), or a doctorate (3.3%).

F. Procedure & Dependant Measures

On game load, each participant was randomly assigned one of three conditions: abstract, vertebrate, or invertebrate. This information, along with each player's current game state was logged on the client-side to ensure participants would never be exposed to another condition, even if they refreshed their browser. Once a participant chose to quit, they were given a post-survey asking about gender, age, country, education, and programming experience. Finally, the survey asked participants to select their level of agreement to the following statements on a 5-level Likert scale to get a sense of participants' attitude towards game elements: 1) "I enjoyed playing the game," 2) "I would recommend this game to a friend wanting to learn programing," 3) "I wanted to help Gidget succeed," and 4) "I enjoyed interacting with the objects in Gidget's world."

After submitting their responses, the participant received a unique code to receive payment for their submission. In addition to the survey responses, we collected a time-stamped activity log for each level a participant attempted including: (1) Each ***press*** of any of the four execution buttons and a copy of the code at the time of execution; (2) ***Level start*** & ***level end***: events marking when a player started and completed or quit a level; (3) ***Idle start*** & ***idle stop***: events marking when a player provided no mouse or keyboard input for 30 seconds or more, and where in the UI panes (Fig. 1) the idle time occurred, including the syntax help pane. Events were also recorded marking resumption in activity; (4) ***Edit time*** (***edit in*** & ***edit out***): events marking when the player clicked inside the code pane (Fig. 1.1) to edit code or clicked elsewhere to leave the editing pane; (5) ***Pane Time*** (***time in*** & ***time out***): timestamps of mouse cursor movement over or out of the seven major UI panes.

From these, we calculated the following dependent measures for each participant: (1) ***Time on level***: how long individual participant was actively engaged with the code and interface of each level overall, adjusted by subtracting ***idle time***. This was calculated for each level by first taking the difference of ***level end*** and ***level start***, then subtracting ***idle time*** for that level; (2) ***Time overall***: how long each participant played the game overall, adjusted by subtracting ***idle time***. This was calculated by summing up the all of the time on level data per participant and subtracting the sum of their ***idle time***.

Finally, we used each participants' *number of levels completed*, *time to complete* or *quit* a level, and logs of *execution buttons* and *UI pane activity*, to compute dependent measures on participants' activity proportional to overall time spent on levels.

IV. Results

Table 4 presents the results for both novice and experienced programmers. We used non-parametric tests, as our dependent

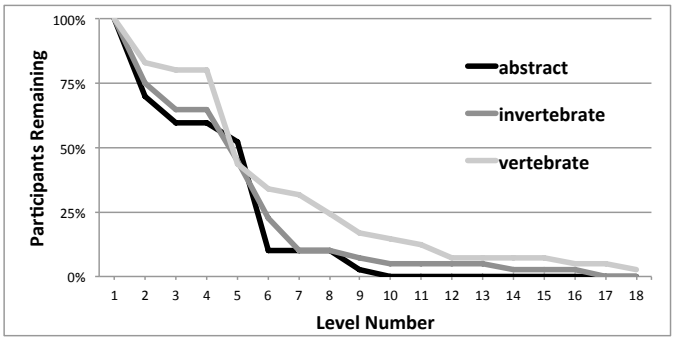

Figure 3. Percent of players remaining for each condition at each level. By level 10, all in the abstract condition had quit.

measures were not normally distributed. Our level of confidence was set at α=0.05.

A. *Vertebrate Condition Players Complete More Levels*

Since we required that all participants complete the first level to receive payment, the minimum number of levels completed was one. The maximum number of levels completed in the abstract, invertebrate, and vertebrate conditions were 9, 16, and 18, respectively. There was a significant difference in the number of levels participants completed between the three conditions ($\chi^2(2,N=121)=7.3,p<.05$). Further post-hoc analysis with a Bonferroni correction shows that the significantly different pair was the abstract vs. vertebrate conditions (W=1380.5,Z=-2.5,p<.01), with the vertebrate group completing more levels. Comparison of the abstract vs. invertebrate (W=1669.5,Z=0.5,n.s.) and vertebrate vs. invertebrate (W=1427.5,Z=-2.0,n.s.) conditions showed no differences.

Investigating this difference further, the distribution of percentage of participants remaining (Fig. 3) shows that approximately 25% of the participants from each group quit the game after completing only the first level. Next, many participants quit on level 4, which required them to use the command learned in the previous level with a new command to complete the goal. Finally, participants quit again in large numbers on level 6, which introduced conditional statements. This is consistent with others' findings that novice programmers have difficulty with conditional logic [9,26]. Here, the abstract condition had the most drastic drop with 90% of its participants quitting, followed by the invertebrate condition's drop of 77.5% its participants, and finally, the vertebrate condition with 67.5% of its participants quitting. All of the abstract condition's participants quit by level 10, whereas both animate, concrete conditions had a few participants complete or nearly complete all the levels.

Since all participants were novice programmers with no statistical difference in demographics, these results suggest that interacting with goals that use concrete data elements had a significant positive effect on participants' engagement with the game, particularly on levels introducing difficult concepts.

B. *Both Animate Conditions Players Play Longer*

There was a wide range of overall play times for the abstract, invertebrate, and vertebrate conditions (4.9 min to 1.3 hrs, 8.3 min to 1.9 hrs, and 6.9 min to 2.8 hrs, respectively). There was a significant difference in the length of time participants played the game overall by condition ($\chi^2(2,N=121)=10.2,p<0.01$). A post-hoc analysis with

TABLE IV. SUMMARY OF NON-PARAMETRIC TESTS

		Novice (χ^2: df=2, N=121)	**Experienced** (χ^2: df=2, N=130)
# of levels completed		($\chi^2=7.3, p<.05$)	($\chi^2=4.3$, n.s.)
Time played overall		($\chi^2=10.2, p<.01$)	($\chi^2=3.5$, n.s.)
Ratio of time played to levels played		($\chi^2=3.7$, n.s.)	($\chi^2=2.9$, n.s.)
Exec	One Step	($\chi^2=2.2$, n.s.)	($\chi^2=0.8$, n.s.)
	One Line	($\chi^2=0.5$, n.s.)	($\chi^2=0.7$, n.s.)
	All Lines	($\chi^2=1.6$, n.s.)	($\chi^2=2.6$, n.s.)
	To End	($\chi^2=0.1$, n.s.)	($\chi^2=0.5$, n.s.)
UI Panes	Code	($\chi^2=2.5$, n.s.)	($\chi^2=0.9$, n.s.)
	Goal	($\chi^2=4.0$, n.s.)	($\chi^2=0.9$, n.s.)
	Execution	($\chi^2=3.7$, n.s.)	($\chi^2=2.7$, n.s.)
	Feedback	($\chi^2=0.3$, n.s.)	($\chi^2=2.8$, n.s.)
	World	($\chi^2=4.3$, n.s.)	($\chi^2=2.1$, n.s.)
	Memory	($\chi^2=1.0$, n.s.)	($\chi^2=0.4$, n.s.)
	Cheat Sheet	($\chi^2=5.8$, n.s.)	($\chi^2=3.4$, n.s.)
	Edit Time	($\chi^2=0.9$, n.s.)	($\chi^2=1.7$, n.s.)

Bonferonni correction reveals that two conditional pairs were significantly different: abstract vs. vertebrate (W=1330,Z=-2.9,p<.016) and abstract vs. invertebrate (W=1889,Z=2.6,p<.016). In both cases, the participants in the abstract condition spent significantly less time playing the game than the other conditions. Play time between the animate, concrete conditions did not differ (W=1620,Z=-0.2,n.s.).

Next, we investigated how quickly players completed levels by comparing participants' ratio of total play time to number of levels played, finding no significant difference ($\chi^2(2,N=121)=3.7$,n.s.). These findings are supported by Fig. 4, which shows that the median times to complete the first 5 levels are comparable across conditions. The sporadic time shown in the remainder of the figure may be attributed to the individual differences of the smaller number of participants completing later levels.

C. *No Significant Differences in Code Execution Strategies*

One possible explanation for the differences in levels completed was a different use of the game UI. Therefore, we investigated the proportion of execution button presses per unit time on completed levels, for each of the four execution buttons, finding no significant differences in usage (*one step*: ($\chi^2(2, N=121)=2.2$,n.s.), *one line*: ($\chi^2(2,N=121)=0.5$ n.s.), *all steps*: ($\chi^2(2,N=121)=1.6$, n.s.), *to end*: ($\chi^2(2,N=121)=0.1$ n.s.)). These results show that the differences in success were likely

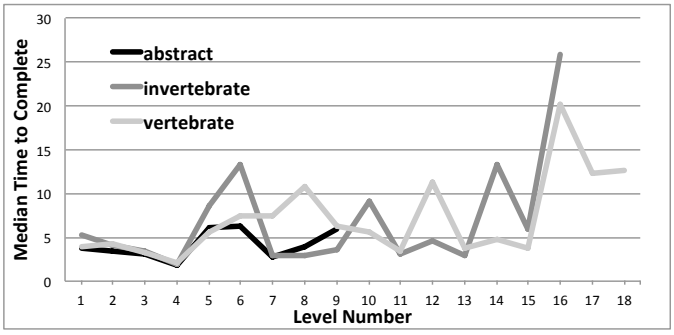

Figure 4. Median times to complete each level, by condition. The times for the first 5 levels were comparable.

not due to one condition executing the program more frequently or stepping through it differently.

D. No Differences in User Interface Usage

Another possible explanation for the disparity in levels completed was differences in how participants used the various panels in the UI. We examined the proportion of interface pane usage to overall time on levels played, again finding no significant differences among conditions (Code: (χ^2(2,N=121) =2.5, n.s.), (Goals: (χ^2(2,N=121) =4.0, n.s.)) Execution: (χ^2(2,N=121) = 3.7, n.s.), Feedback: (χ^2(2,N=121) = 0.3, n.s.), World: (χ^2(2,N=121) =4.3, n.s.), Memory: (χ^2(2, N=121) =1.0, n.s.), CheatSheet: (χ^2(2,N=121) =5.8, n.s.)). We also tested the proportion of time spent editing code (computed from the logs) to overall time on levels and found no significant differences among conditions (χ^2(2,N=121) =0.9, n.s.).

All of these results suggest that the differences in success and play time were not due to different in the proportion of time players used each of the user interfaces in the game.

E. No Significant Differences in Attitudes

Although there was a trend in survey responses indicating a positive experience playing the game, there was no significant difference in participants' self-reported level of enjoyment comparing the three conditions (χ^2(8,N=121)=5.8,n.s.) or whether they would recommend the game to a friend wanting to learn programming (χ^2(8,N=121)=4.1,n.s.). Similarly, there was a positive trend in responses across conditions, but no significant difference in participants' self-reported desire to help Gidget succeed (χ^2(8,N=121)=11.6,n.s.) or whether they enjoyed working with their data elements (χ^2(8, N=121)=5.5,n.s.). Here, our findings are similar to those by Nass et al., where participants did not believe the computer affected their performance, even when it did [29].

F. No Significant Effects for Experienced Programmers

Whereas the rank novices showed significant differences in level completion and play time, players with self-reported programming experience showed no differences in any of the previously reported analyses (Table 4) such as play time or levels completed. This group consisted of 130 participants with 38, 47, and 45 in the abstract, invertebrate, and vertebrate conditions, respectively. These were the participants who reported in the survey that they have: 1) taken a programming course, 2) written a computer program, or 3) contributed code towards the development of a computer program.

V. Discussion

Our findings show that goals involving animate rather than inanimate objects significantly increases learners' engagement in a programming game, leading rank novices to play significantly longer and complete significantly more levels. Moreover, we showed that these effects were not due to differences in how players executed the programs, how they used the game UI, how long they attempted each level, or how much time they spent editing their code. This manipulation had no effect, however, on players with programming experience.

There are several possible interpretations of these results. One possibility is that when reaching a difficult level, players saw a greater purpose in saving an animal or insect than in moving a block, and therefore kept playing when goals were to help animate objects. Prior research on computer science recruitment shows that there are gender specific effects in motivation to enroll, specifically related to the reasons for computing (females are enticed when they see how computing can be used for a purpose) [25]. The non-effect for experienced programmers might be attributed to their familiarity to the concepts being taught, meaning that the effect of the goals was overcome by the lack of challenge.

Another potential explanation is that participants paid closer attention to code involving animate objects and therefore understood the semantics of the programming language better, making the difficult levels easier. This idea is supported by recent work in neuroscience showing that human brains attend preferentially to images of animate over inanimate objects [28]. Additionally, or alternatively, participants in the animate conditions may have been better able to comprehend the robot's explanations of its actions because the messages involved concrete, animate objects that had meaning in the context of the game. Experienced programmers may have been less interested in the robot's explanations; instead, they likely used their prior knowledge of programming syntax and semantics to complete goals, minimizing their attention towards the different data elements' icons and labels.

Because there were no distinguishable differences in play time between levels across conditions, it is also possible that participants had comparable success in learning, but different amounts of retention. In addition to playing a role in learning-related retention [7], the amygdala has been shown to prefer images of animals over other objects [28]; therefore, players may simply have remembered more about the meaning of commands from previous levels when the levels involved animals. The non-effect for experienced programmers can likely be attributed to their familiarity of memorizing and using commands to write computer code.

A. Implications

Our results have many potential implications on our understanding of online learning, the role of game elements in engagement, and computing education pedagogy. Our results show that purposeful goals may play a significant role in engagement in the context of self-guided, discretionary, educational games. These findings support prior works done in classroom settings [19,25], and broadens them to informal learning settings. Future work should investigate the effects of these factors on learning, both in formal and informal contexts.

In addition, this study demonstrated that small changes to the game elements can have a significant effect on engagement in educational games. Here, we had large effect sizes, with double the overall play time and level completion, as was the case in our prior study [22]. This suggests that in the growing amount of work in educational games research, game designers should be doing more on low-level factors that are predicted to be influential by research in learning, memory, and attention.

B. Threats to Validity

Our results have a number of limitations that may restrict their generalizability and validity. First, MTurk allows participants to self-select into HITs. Our HIT did not require any special qualifications and mentioned games and robots, both of which may have led us to recruit participants with interests or experience with both. Although we tried to account for factors that would affect the HIT's listing on Amazon's HIT page, those who filtered for higher-paying HITs would be less

likely to find our HIT, whereas those filtering for a tag labeled "game" would be more likely to find our HIT.

The game was accessible by computer, connected to the Internet, and listed on a website requiring login. Although not directly translatable to programming ability, gaining access to the game required a fair amount of computer skills. Moreover, the exclusion of Internet Explorer (the most utilized browser worldwide at the time of the study [30]) introduced a sampling bias. Our participants were also well educated, with 80.1% reporting they had some college education or beyond.

Capturing a time-stamped activity log of participants' interactions with the game interface is a coarse instrument for measuring attention, particularly when it is done remotely, tracking only mouse and keyboard activity. Although we defined interaction with an interface element as having the mouse cursor over it, it is plausible that users may have been concentrating on other parts of the interface without doing so. This would be acceptable if all the measurements were randomly distributed in the same way across conditions, but could be problematic if they were systematically different.

Finally, though small, there was an economic incentive for participants to participate in the study and to continue completing levels for monetary bonuses. Since these incentives would not exist in classrooms, or in self-guided learning contexts, it is unclear how or findings would generalize to contexts with other forms of intrinsic and extrinsic motivations.

VI. Conclusion and Future Work

By manipulating the visual representation of data elements in the game to influence the purposefulness of game goals, we have found that novice programmers complete more levels and play longer when presented with images of animate, vertebrate animals instead of abstract objects. Our results raise many questions about the underlying mechanisms of this effect, which range from effects on motivation, learning, and attention. In our future work, we hope to investigate these possible mechanisms, leading to a deeper understanding of the effect of the presentation of a computer on a person's ability and desire to program and do so successfully.

References

1. Ainsworth, S.E., Bibby, P.A., Wood, D.J. (2002). Examining the effects of different multiple representational systems in learning primary mathematics. *J. of Learning Sciences*. 11(1), 25-62.
2. Batt, S. (2009). Human attitudes towards animals in relation to species similarity to humans: a multivariate approach. *Bioscience Horizons* 2: 180–190.
3. Bowman, R.F. (1982). A Pac-Man theory of motivation: tactical implications for classroom instruction. *Educational Tech.*, 22(9), 14-16.
4. Bradshaw, J.W.S., Paul, E.S. (2010). Could empathy for animals have been an adaptation in the evolution of Homo sapiens? *Animal Welfare*, 19(1), 107-112.
5. Bruckman, A. (1997). MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids. *MIT Media Lab*.
6. Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. ACM *SIGCSE Bulletin*, 27–31.
7. Chavez, C.M., McGaugh, J.L., Weinberger, N.M. (2009). The basolateral amygdala modulates specific sensory memory representations in the cerebral cortex. *Neurobiology of Learn Memory*, 91, 382–392.
8. Corno, L., Mandinach, E.B. (2004). What we have learned about student engagement in the past twenty years. *Big Theories,* 297–326.
9. Dahotre, A., Zhang, Y., Scaffidi, C. (2010). A qualitative study of animation programming in the wild. *ACM-IEEE ESEM*, 1-10.
10. Downs, JS., Holbrook, MB, Sheng, S., Cranor, L.F. (2010). Are your participants gaming the system? Screening mechanical turk workers. *ACM CHI*, 2399-2402.
11. Freeman, P., Aspray, W. (1999). The supply of information technology workers in the United States. *Computing Research Association*
12. Garris, R., Ahlers, R., Driskell, J.E. (2002). Games, motivation, and learning: a research and practice model. *Simulation & gaming*, 441–467.
13. Goldstone, R., Sakamoto, Y. (2003). The transfer of abstract principles governing complex adaptive systems. *Cognitive Psychology*, 414–466.
14. Hardy, C., Bryson, C. (2010). Student engagement: paradigm change or political expediency? *Networks*, 09, 19-23.
15. Hsieh, G., Kraut, R.E., Hudson, S.E. (2010). Why pay?: exploring how financial incentives are used for question & answer. *ACM CHI*, 305-314.
16. Kaminski, J.A., Sloutsky, V.M., Heckler, A.F. (2005). Relevant Concreteness and its Effects on Learning and Transfer. *Cognitive Science Society*, 1139-1144.
17. Kaminski, J.A., Sloutsky, V.M., Heckler, A.F. (2009). Concrete instantiations of mathematics: a double-edged sword. *J. for Research in Math Edu.*, 40, 90-93.
18. Kearsley, G., Shneiderman, B. (1998). Engagement Theory: a framework for technology-based teaching and learning. *Educational Technology*, 38(5), 20-23.
19. Kelleher, C., Pausch, R., Kiesler, S. (2007). Storytelling Alice motivates middle school girls to learn computer programming. *ACM CHI*, 1455-1464.
20. Kittur, A., Chi, E.H., Suh, BW. (2008). Crowdsourcing user studies with Mechanical Turk. *ACM CHI*, 453-456.
21. Layman, L., Williams, L., Slaten, K. (2007). Note to self: make assignments meaningful, ACM *SIGCSE, 459-463.*
22. Lee, M.J., Ko, A.J. (2011). Personifying Programming Tool Feedback Improves Novice Programmers' Learning. *ACM ICER*, 109-116.
23. Malone, T. W. (1981). What makes computer games fun? *Byte*, 6(12), 258-277.
24. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The scratch programming language and environment. *ACM TOCE*, 1-15.
25. Margolis, J., Fisher, A. (2002). Unlocking the Clubhouse. *MIT Press*.
26. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M. (2010). Learning computer science concepts with Scratch. *ACM ICER*, 69-76.
27. Miller, R.B., Greene, B.A., Montalvo, G.P, Ravindran, B., Nichols, J.D. (1996). Engagement in academic work. *Contemporary Educational Psychology*. 21, 388-422
28. Mormann, F., Dubois, J., Kornblith, S., Milosavljevic, M., Cerf, M., Ison, M., Tsuchiya, N., et al. (2011). A category-specific response to animals in the right human amygdala. *Nature Neuroscience*.
29. Nass, C., Fogg, B.J., Moon, Y. (1996). Can computers be teammates? I*nternational J. of Human-Computer Studies*, 45, 669-678.
30. NetMarket Analytics. Retrieved September 12, 2011, from http://www.netmarketshare.com
31. Oblinger, D. (2004). The next generation of educational engagement. *Journal of Interactive Media in Education*, 8.
32. Oblinger, D., Martin R., Baer, L. (2004). Unlocking the potential of gaming technology. *National Learning Infrastructure Initiative*.
33. Oblinger, D., Oblinger, J. (2005). Educating the net generation. *Educause*.
34. Papert, S. (1980). Mindstorms. *Basic Books*.
35. Ross, J., Irani, I., Silberman, M. Six, Zaldivar, A., Tomlinson, B. (2010). Who are the crowdworkers?: shifting demographics in Amazon Mechanical Turk. *ACM CHI*, 2863-2872.
36. Ryokai, K., Lee, M.J., Breitbart, J.M. (2009). Children's storytelling and programming with robotic characters, *ACM C&C*, 19-28.
37. Vroom, V. H. (1964). *Work and motivation.* New York: Wiley.