In-Game Assessments Increase Novice Programmers' Engagement and Level Completion Speed

Michael J. Lee[1], Amy J. Ko[1], and Irwin Kwan[2]

[1] University of Washington
Information School
{mjslee, ajko}@uw.edu

[2] Oregon State University
School of EECS
kwan@eecs.oregonstate.edu

ABSTRACT

Assessments have been shown to have positive effects on learning in compulsory educational settings. However, much less is known about their effects in discretionary learning settings, especially in computing education and educational games. We hypothesized that adding assessments to an educational computing game would provide extra opportunities for players to practice and correct misconceptions, thereby affecting their performance on subsequent levels and their motivation to continue playing. To test this, we designed a game called Gidget, in which players help a robot find and fix defects in programs that follow a mastery learning paradigm. Across two studies, we manipulated the inclusion of multiple choice and self-explanation assessment levels in the game, measuring their impact on engagement and level completion speed. In our first study, we found that including assessments caused learners to voluntarily play longer and complete more levels, suggesting increased engagement; in our second study, we found that including assessments caused learners to complete levels faster, suggesting increased understanding. These findings suggest that including assessments in a discretionary computing education game may be a key design strategy for improving informal learning of computing concepts.

Categories and Subject Descriptors

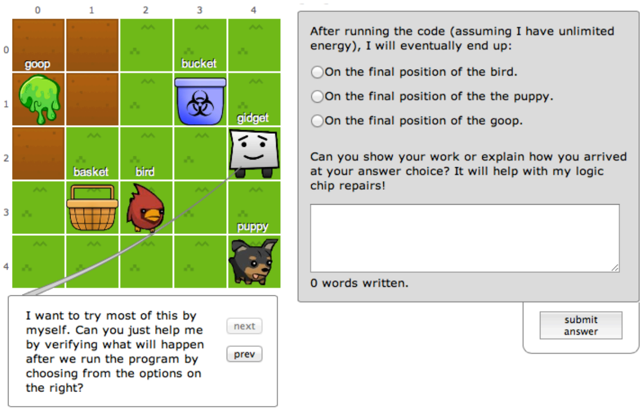
K.3.2 Computer Science Education: Introductory Programming, D.2.5 Testing and Debugging.

Keywords

Programming, assessment, engagement, speed, debugging, serious game, educational game.

1. INTRODUCTION

Recent press about *code.org* and other efforts to increase computing literacy have begun to attract millions of people to learn computer programming. Many of these individuals are turning to discretionary online resources such as Codecademy, Kahn Academy, Coursera, and CodeHS, and research environments such as Alice and Scratch, to learn. Although research on these learning materials is still sparse, learners report that they enjoy these informal resources more than traditional classes because they allow for flexibility in how they learn, they give learners a better sense of retaining the material [5], and they are more motivating, engaging, and interesting than traditional classroom courses [10]. Some of these attitudes can be attributed to these resources' use of game mechanics such as scaffolded materials, structured mastery learning, concrete goals, and extrinsic incentives such as badges [39].



After running the code (assuming I have unlimited energy), I will eventually end up:

○On the final position of the bird.

○On the final position of the the puppy.

○On the final position of the goop.

Can you show your work or explain how you arrived at your answer choice? It will help with my logic chip repairs!

0 words written.

I want to try most of this by myself. Can you just help me by verifying what will happen after we run the program by choosing from the options on the right?

Figure 1. Does providing in-game assessment questions help discretionary learners playing an educational programming game increase engagement and level completion speed? This figure shows a multiple choice assessment in such a game.

Unfortunately, many of these resources struggle to keep learners engaged [12] and few of them involve explicit evaluations of learning, making it unclear how much learners actually learn or retain. Therefore, as these resources increase in popularity, a significant design challenge will be improving engagement, while also demonstrably improving understanding.

One way to potentially improve both understanding and engagement is to use assessments [29]. Assessments, which directly tests learners' knowledge by asking them to explicitly answer questions about the material, are widely used in compulsory settings not only to measure learners' progress and what they know [6], but also to improve students' learning itself [4]. Assessments improve learning and understanding partly by helping students practice course material and by clearing up misconceptions [8,20].

Unfortunately, there is a lack of research about how including assessments might affect learners' use of discretionary learning resources [5]. Moreover, there is reason to believe that assessments could actually harm engagement, even if they improve learning. For example, assessments can lead to test-anxiety, negatively affecting engagement [34], especially if they get the wrong answer or feedback is lacking [6]. Including assessments in educational games or resources that use game mechanics may be even more harmful, as they may interfere with a player's enjoyment of the game, creating a "testing" mode that is poorly integrated with the rest of the game, leading the learner to disengage or even quit the activity.

To begin exploring the role of assessments in discretionary computing education games, we investigated the effect of integrated learning assessments on both *engagement* and *speed* across two online controlled experiments where learners played Gidget [22,23], a debugging game in which learners play through a series of levels,

ICER'13, August 12–14, 2013, San Diego, California, USA.
Copyright © 2013 ACM 978-1-4503-2243-0/13/08…$15.00.
http://dx.doi.org/10.1145/2493394.2493410

finding and fixing defects in a robot's programs. In our two experiments, we manipulated the inclusion of explicit assessments like the one shown in Figure 1, which asks learners to indicate the final position of the robot by mentally simulating the given program's execution. In the rest of this paper, we discuss prior work on educational games and assessments in different learning environments, detail our game and study design, and discuss our results and their implications on computing education.

2. RELATED WORK

Educational games have become an increasingly popular way to teach STEM (science, technology, engineering, and mathematics) subjects [16,30]. Researchers have taken advantage of this interest to improve educational games to be more fun, informative, and educational. Some research focuses on creating games that directly try to teach a skill or subject such as computer programming [13,22,23], others focus on adding game-like features to existing teaching systems such as intelligent tutors [18,26], and some focus more generally on creating frameworks for effective evaluation [2,34,35]. Several works have also attempted to identify the specific parts of games that motivate [24] and attract people to pursue computing education [14,15,24], but fewer have examined how to effectively measure the outcomes of educational games [35].

Assessments have long been used in compulsory settings to keep students' engaged with the course material, encourage reviewing or reflecting on past material, and measure learning [4,33]. They have been shown to be useful for students to orient, organize, and integrate study materials, leading to measurable performance gains [7]. Moreover, receiving feedback from assessments can affect learners' self-efficacy, which plays an important role in future performance [38], and also provides an opportunity for learners to reflect on their work and correct any misconceptions they have about the material [8,20]. Of particular interest is the *testing effect*, which states that minor, incremental testing leads to better retention and higher scores when compared to repeatedly studying the same material [7,17,25]. This demonstrates that assessments not only have utility for educators to track their students' progress, but are also important tools to promote student learning and retention.

Much less is known about assessments in discretionary settings, particularly for educational games, and how they might interact with other gameplay elements to affect players' engagement with the educational material. This is particularly true of learning technologies for computing education. Although researchers agree that adding assessments to games is worthwhile since they build on the same principles of learning (to be successful in the game, or to learn some additional material) [27], the lack of use in actual practice makes it unclear how the inclusion of assessments in games will influence players. This is a challenging task, since deviating too far from the gameplay can be distracting, boring, or disengaging, which is particularly important in the context of a discretionary setting where a learner can choose to leave at any time [34]. Csikszentmihalyi describes this as maintaining flow, where the learner is deeply immersed in the game experience [11]. Shute et. al's work attempts to address these issues through stealth assessments, which are intended to reduce text anxiety, and blur the distinction between game content, assessment, and learning [34,35]. The goal of this method is to reduce the likelihood of explicit assessments from disrupting flow and decreasing engagement while playing a game. However, research in compulsory settings, where students know they are being tested, have found that explicit assessments and feedback help students retain information better and keep them motivated with the educational material [7,17,25]. Therefore, it is currently unclear how making assessments an explicit part of a game will affect learners' perceptions of the game, their motivation to play, their recall ability, and their performance.

3. METHOD

The aim of our study was to determine how integrated, explicit assessments in an educational computing game affects engagement and task completion speed in self-directed learners, and to identify the extent of these effects. To do this, we designed Gidget (Fig. 2), a game that teaches programming by asking the player to fix a damaged robot's faulty code to help it accomplish its missions (detailed further in Section 3.1). Each game level teaches a particular programming concept, challenging the player to find and fix the defects in each level's program so that it passes the provided goals, which are executable test cases. Following the mastery learning paradigm [28], each of the game's levels is designed to be passable only if the learner has grasped a particular concept in the game's programming language.

We conducted two separate controlled experiments, with the first measuring engagement and the second measuring task completion speed (see Table 1). Each of our experiments had two conditions: the *control* condition's curriculum consisted of a series of levels without assessments, whereas the experimental condition (which we will call the *assessment* condition), was identical, but also included two assessment levels at the end of each set of levels.

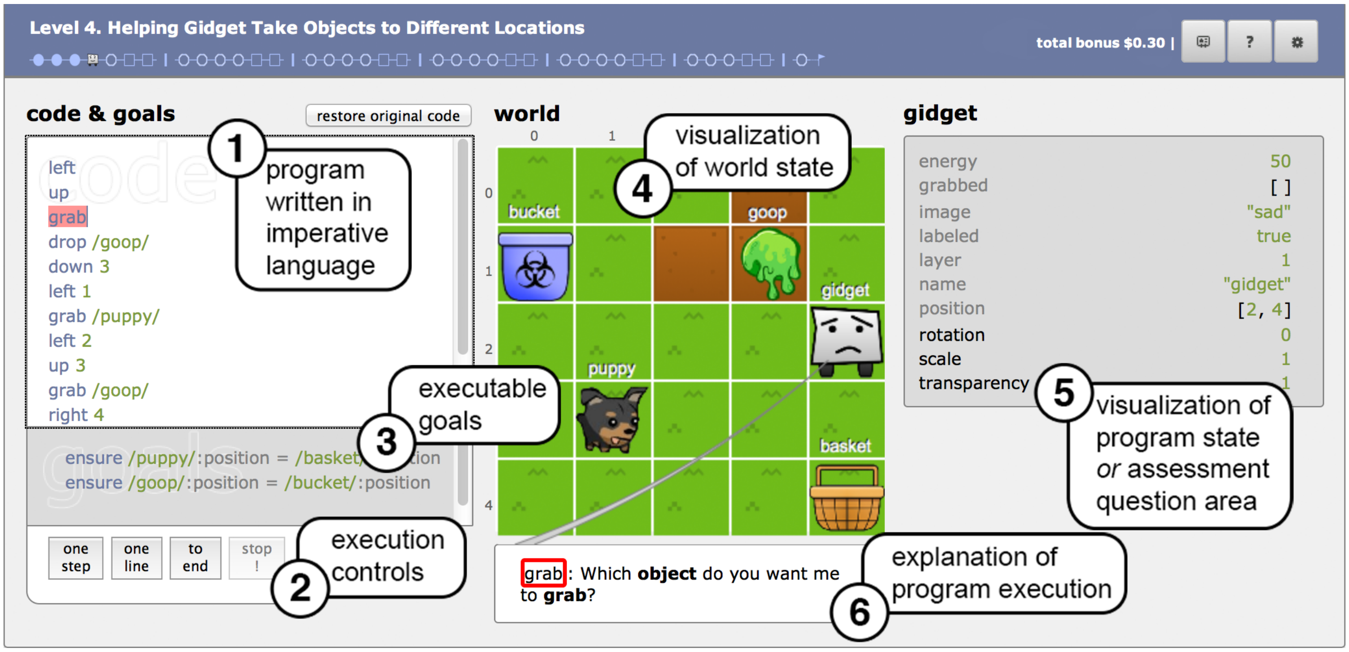
In our *engagement* study, learners could quit any time, as with any discretionary learning material. We hypothesized that the learners in the assessment condition would play the game for longer and complete more levels because the assessments would offer additional opportunities to practice each units' concepts, leading to better understanding of the material and reducing the likelihood of encountering difficulties and discouragement in subsequent levels. We measured both the total number of levels completed and the total time playing the game.

Our *speed* study followed the same structure as the *engagement* study but was designed to enable a direct comparison of how quickly participants completed Gidget game levels. To enable this comparison, we operationalized *speed* as the total time required to complete the first three sets of levels in the game. We hypothesized that even though players in the assessment condition would have to spend more time on the assessment levels, the extra practice and feedback they received through the assessments would result in them being more successful in subsequent levels, completing individual levels faster than those in the control condition.

Both the *engagement* and *speed* studies were between-subjects designs with 200 and 30 participants respectively, split evenly across conditions. Participants were recruited on Amazon.com's Mechanical Turk. The *speed* study was launched after the *engagement* study, without overlap, to prevent people from playing the game simultaneously; we also prevented players more than once. Though we attempted to recruit the same number of participants for both treatments, the *speed* study attracted fewer participants because it required a larger up-front time commitment.

Table 1. Experimental design of our two studies.

	Study 1: "Engagement"	Study 2: "Speed"
Independent variables	Gidget game with or without assessment levels	
Dependent variable(s)	#1: the number of regular levels completed #2: total time playing the game	#1: the time required to complete a set of levels (3 units of the game)
Participants	200 total; 100 per condition	30 total; 15 per condition
Payment criteria	Incentive of $0.10 per level completed, up to a total of 25 (control) or 37 levels (assessment); can quit playing at any point.	Incentive of $7.00 after playing 3 units worth of levels; can quit any time after 3 units, but earn additional $0.10 per level completed.


Level 4. Helping Gidget Take Objects to Different Locations
total bonus $0.30 |
?
code & goals
restore original code
world
gidget
0
1
0
1
2
4
bucket
goop
gidget
puppy
basket
one step
one line
to end
stop !
1 program written in imperative language
2 execution controls
3 executable goals
4 visualization of world state
5 visualization of program state *or* assessment question area
6 explanation of program execution

```
left
up
grab
drop /goop/
down 3
left 1
grab /puppy/
left 2
up 3
grab /goop/
right 4
```

```
ensure /puppy/:position = /basket/:position
ensure /goop/:position = /bucket/:position
```

energy	50
grabbed	[ ]
image	"sad"
labeled	true
layer	1
name	"gidget"
position	[2, 4]
rotation	0
scale	1
transparency	1

grab: Which **object** do you want me to **grab**?

Figure 2. Gidget is a game in which players help a damaged robot by fixing its broken programs. The player's progress is shown below the level title, with learning units divided by vertical bars. Regular levels are circles, and assessment levels are squares.

3.1 Gidget, The Programming Game

Gidget is an HTML5 web application that is playable in a browser. The game is motivated by a story: there has been a chemical spill from a factory and Gidget, a small robot capable of identifying and solving problems with programs, has been deployed to clean up the area. Unfortunately, Gidget was damaged in transit, and is only able to provide code (Fig. 2-1) that partially, but not completely solves each level's goals (Fig. 2-3). It is the player's job to help the robot through missions by diagnosing and fixing the problems in each level's code (Fig. 2-1), then executing the code (Fig. 2-2), so the robot can fulfill the executable mission goals (Fig. 2-3).

The game uses an imperative, Python-like programming language designed specifically for the game. The language supports dynamically typed-variables, Boolean operators and expressions, conditionals, mathematical operators, objects, functions, and domain-specific keywords for the game characters to interact with their world. These interactions primarily include finding things in the world (Fig. 2-4), going to them, checking their properties, and carrying them to other places on the grid. In some cases, objects have their own abilities, which Gidget can call as functions. After each execution step, the effect of these commands are shown in the 'program state' panel (Fig. 2-5) and explained by Gidget (Fig. 2-6) to reinforce the semantics of each command. Each step costs Gidget 1 unit of '*energy*' (displayed at the top of Fig. 2-5), which forces players to consider how to write efficient programs that can be solved using the allocated amount of energy.

To aid the players with debugging, the game offers four execution controls: *one step*, *one line*, *to end*, and *stop* (Fig. 2-2). The *one step* button evaluates one compiled instruction, displaying text explaining the execution of the step. The *one line* button evaluates all steps on one line of the code, just as a breakpoint debugger does, jumping to the final output of that line. The *to end* button evaluates the entire program and the goals, animating each step in quick succession. The *stop* allows the player to halt the program and edit code during any part of the execution. When the learner uses *step* or *one step*, Gidget provides a detailed explanation of the execution of each statement in the program, highlighting changes in the runtime environment. This serves as the game's primary instructional content, explicitly teaching the language syntax and semantics.

To help learners start playing Gidget, the game presents a ten-slide tutorial to every player upon game start. The game also features an in-game reference guide, providing explanations and examples of each command in the language, along with information about programming concepts such as variables, functions, the stack, and loops. The reference guide was available as a standalone help guide or as tooltips that appeared when hovering over tokens in the code editor. Finally, the game's code editor provides keystroke-level feedback about syntax and semantics errors, as in Fig. 2-6, highlighting erroneous code in red and explaining the problem in Gidget's speech bubble.

3.2 Curriculum

There were a total of 7 units in the game, with each unit containing a set of levels focusing on a related set of programming keywords or concepts (see Fig. 3). Unit 1 focused on moving Gidget and other objects around in the world by using simple keywords such as up, down, left, right, grab, and drop. Unit 2 furthered the ideas from the previous section, introducing the goto keyword, and working with lists. Unit 3 introduced variables, types, and values. Unit 4 presented the declaration and use of functions and objects. Unit 5 showed how to use Boolean values, expressions, and logic. Unit 6 focused on loops. Finally, Unit 7 did not teach any new concepts, instead challenging the player to write solutions from scratch to satisfy the level's goals. The last levels in each unit were designed to be a cumulative overview, requiring the learner to recall and use the keywords and concepts covered in that unit.

The order of units and the sequence of levels was designed iteratively based on curricula found in CS1 textbooks, pilot testing with novice programmers, and the authors' cumulative experience teaching CS1 courses. A list of overall learning objectives drove the creation, consolidation, and refinement of the levels [2]. Each level was designed to address one or two specific learning objectives related to the language syntax or semantics. The sequence of levels was also influenced by the game story, and by the language itself (since certain keywords and concepts are easier to understand once

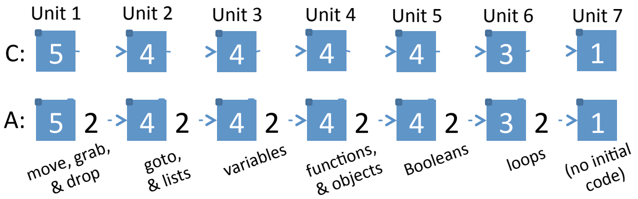


Figure 3. The level sequence for the (C)ontrol and (A)ssessment conditions. The dark boxes show the number of regular levels and the light boxes show the number of assessment levels, each grouped into units.

other concepts have been learned). Finally, levels were designed to progress the story and have some purpose embedded in the goals since we found previously that purposeful goals influence learners' engagement [23]. This sequence was validated by testing with participants in-person and online by observing that the order of levels was not a barrier in their progress through the game.

3.3 Assessment Levels

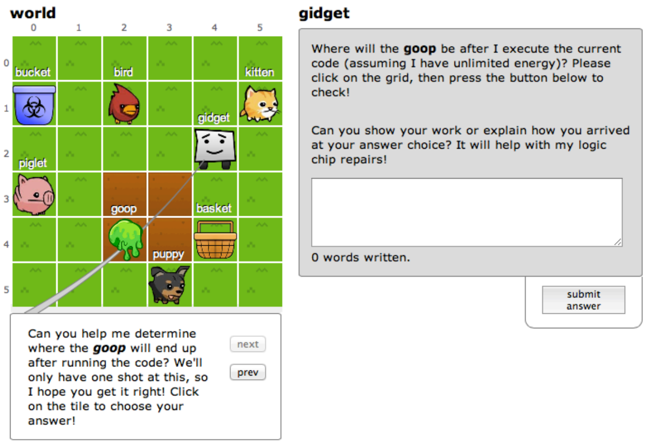
The primary manipulation in our two studies was the inclusion or exclusion of assessment levels in the game. Control condition learners played the game without assessment levels for 7 units, spanning a total of 25 levels (Fig. 3). In contrast, assessment condition learners played the game with two assessment levels at the end of each unit (except the final unit) for a total of 37 levels (Fig. 3). Other than the inclusion of these assessment levels, the sequence and content of the levels were identical in both conditions.

The assessment levels were framed in a way to flow with the story and encourage learners to help the robot with repairs to its logic chip. We took extra care to ensure that the assessments were as close as possible to other game levels, using the same interface, but disabling the code editor, code execution buttons, tooltips, and reference guide, requiring learners to recall their knowledge from the previous levels, much like an exam would in a classroom setting. Related studies have found that "closed book" exams demand more difficult and intricate retrieval mental processes, but also amplify testing effects [3,19]. Gidget explained these constraints by stating the desire to complete the assessment levels using minimal help from other resources.

Assessment levels came in two varieties: multiple choice (Fig. 1), and click-on-the-grid (Fig. 4). Multiple choice assessments (Fig. 1) required the player to select from one of the provided options, which were randomized to minimize ordering effects [21]. All multiple choice questions had one correct key, and three or four incorrect distractors. Click-on-the-grid assessments (Fig. 4) required the player to select a grid location as their answer to level question which asked where either Gidget or another object in the world would be located after the given code was run. The number of possible choices were equal to the number of grid tiles for the level.

In addition to requiring the selection of a multiple choice option or grid location, learners also had to write an explanation of 8 words or more explaining their reasoning before submitting their answer. This self-explanation approach has been shown to minimize guessing and contribute to students' learning and understanding [9,36].

Both types of assessments required learners to inspect the grid, program, and goals, and then mentally simulate the execution of the program to determine the intermediate or final state of some object in the game world. These were therefore direct assessments of players ability to precisely and accurately reason about the language semantics. Clicking the "submit answer" button ran the code, step-by-step, visually showing the player how the code was being processed, and the final state of the program. Gidget would then check the learners' answer choice. If the choice was incorrect, Gidget would show a sad face, give an explanation about why it was wrong, then, show a happy face and proceed to explain what the correct answer was, and why (Fig. 5, left). If the answer choice was correct, Gidget would show a happy face and explained why the learners' answer choice was correct (Fig. 5, right). These design decisions were based on our prior study that found personified feedback affected learners' engagement in a game [22], and studies in classroom settings that show immediate feedback for exam questions enhances retention of the tested materials and reduces negative effects by incorrect answer choices or distractors [6].



Where will the **goop** be after I execute the current code (assuming I have unlimited energy)? Please click on the grid, then press the button below to check!

Can you show your work or explain how you arrived at your answer choice? It will help with my logic chip repairs!

0 words written.

Can you help me determine where the ***goop*** will end up after running the code? We'll only have one shot at this, so I hope you get it right! Click on the tile to choose your answer!

Figure 4. An assessment level where the learner has to click where a particular object will be on the world grid after the level's program is run.

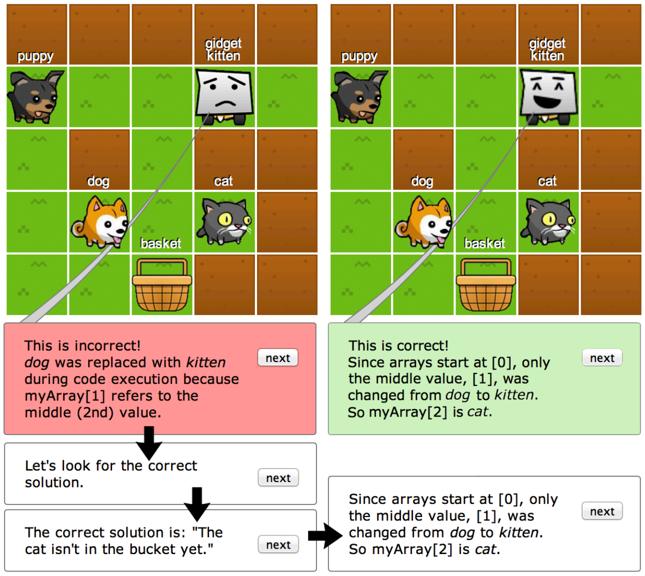
The content of the assessments were designed to test the specific ideas, concepts, and syntax rules covered in each unit. Distractors were designed deliberately to test for common programming misconceptions. Unit 1's assessments were designed to be straightforward so that learners could get familiar with how the assessment levels worked. Unit 2's assessments identified if learners could follow the control flow and use the correct syntax for list queries. Unit 3's assessments tested variable assignment and accessing array values correctly by index. Unit 4 tested variable passing to functions and objects. Finally, Unit 5 tested whether the learners could correctly trace control flow through conditional statements. Like our curriculum, all assessment levels were validated with participants in-person and online by observing that they were sufficiently challenging, that they covered the concepts from our list of learning objectives, and that they were not a barrier in progressing through the game.

3.4 Participants and Procedure

We targeted non-programmers, defined as individuals who self-reported that they had never written computer code and had never taken a course related to computer programming. As mentioned in Section 3, 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).

Our pricing model and validation method was primarily carried over from two prior studies [22,23]. 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 to have any bonus payments to have a minimal effect on a worker's decision to continue playing.

For our *engagement* study, where learners could quit at any time after the first level, we set our base reward as $0.30 for starting the HIT, and an additional $0.10 for each level completed. We set the ceiling for submission time to 5 hours so that participants could gauge the difficulty of the HIT compared to other HITs.



This is incorrect!
dog was replaced with *kitten* during code execution because myArray[1] refers to the middle (2nd) value.

Let's look for the correct solution.

The correct solution is: "The cat isn't in the bucket yet."

This is correct!
Since arrays start at [0], only the middle value, [1], was changed from *dog* to *kitten*.
So myArray[2] is *cat*.

Since arrays start at [0], only the middle value, [1], was changed from *dog* to *kitten*.
So myArray[2] is *cat*.

Figure 5. Example feedback from the game assessments, where the *left* image shows the sequence of messages when the learners' answer is wrong, and the *right* shows the messages when the answer is correct.

For our *speed* study, players were required to complete the first three units to receive payment (totaling 13 or 19 levels, depending on the condition). There were no similar HITs to base our payment on, so we ran several pilot tests to determine an optimal payment rate. The HIT description was identical to that of the engagement study, but also included text explaining that players were required to complete "half the game" before being allowed to quit, and that it could take several hours based on our past observations. We found that nobody accepted/completed our HIT until we started paying $7 to complete the first half of the game and $0.10 for each additional level completed (interestingly, *engagement* study participants would have only been paid $2.20 to complete the same number of 19 levels, or a maximum of $4 for completing the entire game).

On game load, each participant was randomly assigned to the control or assessment conditions. This information, along with their current state in the game were logged on the client-side to ensure participants would not be exposed to the other condition, even if they refreshed their browser. Once a participant chose to quit, they were given a survey to collect demographic data (e.g. gender, age, education) and a unique code to receive payment for their submission. In addition to the survey responses, we automatically collected the number of levels completed, timestamps for level start, level completion, quit, all character-level edits to each level's program, and execution button presses.

Each of our studies were between-subjects, with an even split between the two conditions. Demographic data revealed that participants in both studies and conditions were well proportioned, with no significant differences between groups by gender, age, or education (see Table 2). Consistent with other studies about the demographics of MTurk workers [32], we found that our participants were well-educated, with the majority reporting that they had at least some college education or beyond (Table 2).

4. RESULTS

We provide quantitative evidence for our hypotheses about engagement and speed. Throughout this analysis, we use the non-parametric Wilcoxon rank-sum test with α=0.05 confidence, as our data were not normally distributed.

Table 2. Participant demographics.

	Study 1. Engagement		Study 2. Speed	
	control n=100	assessment n=100	control n=15	assessment n=15
gender	55 males 45 females	58 males 42 females	9 males 6 females	8 males 7 females
some college	86%	87%	93%	100%
age	18-57 years med=27.5	18-64 years med=26	21-40 years med=29	19-36 years med=26

4.1 Engagement: Effect on Level Completion

One of our measures of engagement was the number of levels completed (see Table 3). To enable comparison of how far learners had progressed through the game's instructional content, we subtracted the number of assessment levels completed from the total number of levels completed (see Table 3).

There was a significant difference in the number of non-assessment levels completed between the control and the assessment conditions (W=10851, Z=1.97, N=200, p<.05). Participants in the assessment condition voluntarily completed a median of 8 levels, whereas the control condition completed a median of 6 levels. As additional confirmation, we identified that within the speed study, assessment condition participants were more likely to continue playing the game past the minimum required 3 units, voluntarily completing significantly more levels than the control condition participants (W=277.5, Z=2, p<.05).

We examine more closely what may have influenced a participant's decision to stop playing in Fig. 6. Everyone completed at least 2 levels and 1 control condition participant and 4 assessment condition participants completed the entire game. Many participants from both groups quit the game after completing level 5 (10 players in the control, 15 in the assessment) and level 6 (28 players in the control condition, 12 players in the assessment condition). Level 6 corresponds to the beginning of a new unit (in this case, starting the *goto & lists* unit), and Level 7 required learners to combine the use of keywords from the previous unit and the new unit. Next, 21 of the control group players quit after level 9 (level 10 started the *variables* unit), and 11 assessment group players quit after level 13 (level 14 began the *functions & object*s unit). Since participants had little programming knowledge and there was no difference in demographics, the assessments likely affected motivation when new concepts were being introduced (i.e. starting a new unit) and when they had to be combined with previously learned concepts.

4.2 Engagement: Effect on Play Time

Our other measure of engagement was total time played. After subtracting the time played in assessment levels, we found that participants in the engagement study's assessment group voluntarily played the game for significantly more time than participants in the

Table 3. Summary statistics for studies and conditions. ("adj" = adjusted to exclude assessment levels).

	Study 1. Engagement		Study 2. Speed	
	control n=100	assessment n=100	control n=15	assessment n=15
Min. levels completed	2	2, 2 (adj)	13	19, 13 (adj)
Median levels completed	6	10, 8 (adj)	14	19.5, 14 (adj)
Max. levels completed	25	37, 25 (adj)	25	37, 25 (adj)
Min. time played	6.9 min	6.8 min	57.1 min	63.4 min
Median time played	26.3 min	41.9 min	121 min	102.2 min
Max. time played	142 min	296 min	188.6 min	198.3 min

control group (W=8434.5, Z=-3.9, N=200, p<.01). As shown in Fig. 7, the assessment condition learners voluntarily played twice as long as those in the control condition, with a median overall play time of 41.9 minutes and 26.3 minutes, respectively.

Combined with the large difference in levels completed described in the previous section, the significant differences in play time suggests that assessments caused learners to continue playing even when reaching unit boundaries or difficult levels.

4.3 Speed: Effect on Level Completion Time

While the engagement study results show that participants stayed engaged longer when given assessments, this effect could be due to either improved motivation, improved understanding, or a combination of the two. To separate these effects, our speed study held the incentives constant, requiring every participant to complete a minimum number of levels for compensation.

Table 3 shows the descriptive statistics for the speed study results. We found no significant difference in the total time participants played the first three units of the game (W=222, Z=-0.4, N=30, n.s.), even though learners in the assessment condition were required to play an additional six levels. However, if we adjust the times by excluding the time spent on assessment levels, we find the difference in completion time was significant (W=171, Z=-2.5, N=30, p<.05), with participants in the assessment condition completing the three modules *twice* as fast overall, compared to control condition learners. This shows suggests that assessments helped learners master the game's concepts faster and that adding a small number of assessment levels essentially costs no extra time for participants, but leads to better performance and engagement.

4.4 Speed: Effect on Play Time & Style

To better understand how participants used their time playing the game in the speed study, we examined participants' code versions, code executions, and code edit time. Descriptive statistics for all the data reported in this section can be seen in Table 4.

There was no significant difference in the overall number of code versions participants ran for the first three units between conditions. In addition, there was no significant differences in how frequently the participants used the incremental execution control buttons (*one step, one line,* and *stop*) overall for the first three units of the game. However, participants in the control condition used the *to end* execution button significantly more than their counterparts, suggesting that they consumed significantly less instructional content, as the *to end* execution prevented players from reading Gidget's explanations of program execution. Control condition participants also spent (nearly significant) more time editing their code (as indicated by having their mouse cursor or text caret in the coding pane) than those in the assessment condition. These results suggest that learners in the assessment condition may have spent more time understanding program semantics by executing the program stepwise instead of reading it or executing it at full speed.

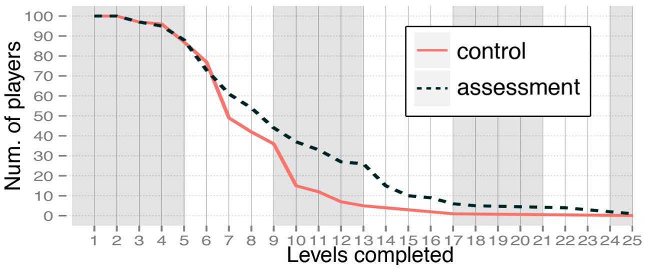


Figure 6. The *speed* study's *adjusted* levels completed, showing how many participants remain after each level. The vertical bars indicate unit boundaries.

Inspecting the overall time each participant spent on each unit, we found a general trend of assessment condition participants completing levels faster than the control group (see Table 4, median times for last 3 rows). This is especially true of the third unit, which shows that the assessment condition participants completed the unit significantly faster than their control counterparts. Closer examination shows that participants in the assessment condition finished significantly faster in the first level of the third module (W=164, Z=-2.8, N=30, p<.01), which introduced variables, and the fourth level of the third module (W=172, Z=-2.5, N=30, p<.05), which required the use of all the keywords and concepts used throughout the unit.

Finally, we calculated how much time assessment condition participants spent on assessment levels in relation to regular levels. Overall, they played a median of 22 minutes across 6 assessment levels. Checking the ratio of assessment play time to overall play time, we found that participants spent a median of 23.8% of their total time playing assessment levels.

We also examined how well assessment condition learners performed on the assessments and found that they averaged 4 out of 6 correct. We read through participants' incorrect responses to identify their misconceptions. We found that the majority of misconceptions were the ones we expected and for which we created appropriate distractors (as detailed in Section 3.3). The one misconception that learners encountered that we had not expected was in the first unit. In the first assessment level, 6 learners were unsure whether a number was required after a move command (e.g., "up" vs. "up 1") and got the answer incorrect. However, 5 of these 6 participants were able to correctly answer the next assessment, which asked a similar question. We suspect that misconceptions here and in later assessment levels were addressed and clarified since each assessment showed the code execution, explained why the participants' chosen answer was false, and why the correct answer was true even if the participants' answer choice was correct.

5. DISCUSSION

Our findings demonstrate that including explicit multiple choice assessments with self-explanations in a discretionary programming game can significantly increase learner's engagement and speed. In the case of Gidget, these effects were strong, with the learners given assessments completing 30% more non-assessment levels (Table 3), playing twice as long (Table 3), and completing levels about 20% faster (Fig. 7), than those not given assessments.

We also found that speed study learners in the assessment condition were only getting the correct answer 66.67% of the time. However, they were spending an average of 24.9% of their total play time on the assessments, and in the free-form answer section, many participants gave reasonable explanations for why they thought their particular answer was correct (see Section 4.4). This indicated that learners were trying on the assessment levels, even though they could proceed regardless of the outcome of their answer.

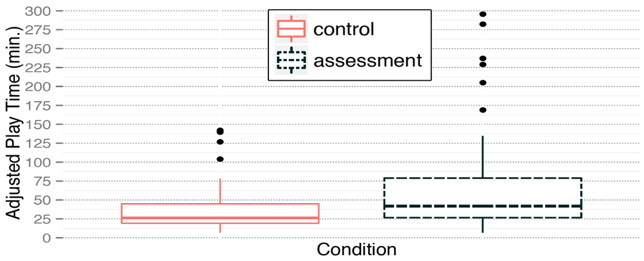


Figure 7: In the *engagement* study, players in the assessment group voluntarily played the game for significantly more time than players in the control group.

Table 4. Summary statistics for the *speed* study learners' play styles, with significant results in bold.

	control n=15	assessment n=15	significance test N=30	
code versions	58-223 med=75	50-124 med=71	W=204.5, Z=-1.1	n.s.
"one step" clicks	0-2901 med=268	1-1883 med=256	W=243, Z=0.4	n.s.
"one line" clicks	8-426 med=90	0-357 med=40	W=243, Z=0.5	n.s.
"to end" clicks	**11-73** **med=31**	**4-59** **med=19**	**W=166.5, Z=-2.7**	**p<.01**
"stop" clicks	0-112 med=13	1-51 med=21	W=241.5, Z=0.35	n.s.
focus time	**29.1-177.2sec** **med=61.7**	**22.7-105.2sec** **med=39.6**	**W=185, Z=-1.9**	**p=.051**
unit 1 completion	10.8-57.2min med=22	7.6-46.9min med=24.2	W=227, Z=-0.02	n.s.
unit 2 completion	12.1-70.2min med=37.3	20.1-73.9min med=33.5	W=210, Z=-0.9	n.s.
unit 3 completion	**8-108min** **med=40.8**	**6.4-110.4min** **med=19.8min**	**W=176, Z=-2.3**	**p<.05**

There are several possible interpretations of our results. The difference in performance that we saw in the *speed* study might be because the assessment levels corrected misconceptions by providing the correct answers (whether or not the learner submitted the correct answer), which has been shown to improve performance in compulsory settings [8,20].

It is also possible that since the code in assessment levels were not executable, learners had to mentally simulate and trace the program's execution, giving them practice understanding the program semantics unaided. Since control condition leaners' were never presented levels with these constraints and were always able to execute their code to see what happens, they were likely more inclined to do that; this is consistent with the finding that control condition learners used the "to end" execution button significantly more than the assessment condition learners (see Section 4.4), rather than taking the time to understand how the code was running.

A third explanation is that since access to the reference guide and tooltips were disabled in assessment levels, learners in the assessment conditions were required to recall how the keywords and syntax worked using their memory. This may have allowed them to understand both the syntax and semantics of the keywords better than those in the control condition, who were always presented with levels that allowed quick access to definitions and examples through the tooltips and reference guide.

Finally, since assessment levels required an explanation of the answer choice, assessment condition learners had extra opportunity to reflect on their selections and translate that into text. This may have allowed them to identify misconceptions on their own, and may have also provided a means of direct comparison to the answer explanations Gidget gave about the incorrect and correct answer choices. Allowing learners to explain their answer choices may also increase the positive effect of assessments [1], including improved understanding of the material being assessed [9].

These observations may also explain the completion of more levels and the longer play time by the assessment group in our *engagement* study. Those in the control condition may have found later levels too difficult, causing frustration and ultimately making them quit. In contrast, those in the assessment condition may have found these levels less difficult due to their experience from assessment levels, but sufficiently challenging to keep them engaged with the game.

These results have implications for future work in educational games and other discretionary computer programming education resources. In our study, we found that our learners completed more levels, played the game longer, and were faster in regular levels when given assessments. Integrating assessments in a manner that flows with a game's interactions, is part of the story, and framed in a way that has the player helping a game character in a fun way, appears to keep learners engaged, even when these tasks are still obviously a test. Codecademy could integrate assessments that are tightly coupled with the style and content of the site. Not only can it improve students' learning, but it can also be used as a resource to track students' progress and make adjusting changes or suggestions to their curricula. Our findings may be less generalizable to large-scale resources like massively open online courses (MOOCs) however, which are still largely similar to traditional classroom settings, where there are lecture/content resources, and assignments. However, moving MOOCs' content to more interactive instruction and assignments may open the opportunity to integrate assessments in a manner similar to the study we have described.

5.1 Threats to Validity

Our study has several limitations that limit its generalizability. First, MTurk allows participants to self-select into HITs given that they meet certain qualifications. Our HIT only required that participants were living in the USA and had no programming experience. Additionally, filtering HITs for certain payouts or tags could have affected participant recruitment. These limitations introduce a sampling bias, which may limit the generalizability of our results to the particular populations found on MTurk.

There may also be limitations to the generalizability of the game itself. We focus on a specific type of programming language and a specific framing of programming. These may have interacted with assessments in a way that may not occur in other settings.

Finally, though small, there was an economic incentive for participants to participate in the study. We tried to minimize this effect as much as possible, and the feedback from of our pilot study participants and the majority of MTurk participants suggests that people would be willing to play the game without these economic incentives, especially if they were allowed to quit at any time.

6. CONCLUSIONS & FUTURE WORK

We investigated whether providing assessments to self-directed, independent learners playing a game designed to teach programming would increase a learner's engagement and speed in the game. By adding assessments to the end of each unit—a collection of levels designed to teach a set of programming keywords or concepts—we found that learners who were given assessment levels complete more game levels and play the game longer and that they completed non-assessment levels faster.

These findings raise many questions for future work. How important was it that the assessments be tightly integrated into the game mechanics, and what is the effect of other instructional content on engagement and speed, such as worked examples [37] and adaptions of peer instruction [31]? How do these effects play out in other forms of discretionary computing education such as MOOCs, online tutorials, and constructionist approaches to learning programming? With the rising interest in learning to program, and the proliferation of resources to do so, we believe that knowledge about the interaction between the design of these resources and engagement and learning will be essential for learner retention and effective pedagogy.

7. ACKNOWLEDGEMENTS

We thank Philip Reed for contributions to the in-game reference guide, Andre Stackhouse for his help with levels, and Sean Fullerton for discussions about assessments. This material is based upon work supported by the National Science Foundation (NSF) under Grants CNS 1240786, CCF 0952733, and OISE 1210205. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

8. REFERENCES

1. Aleven, V., & Koedinger, K.R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based cognitive tutor. *CogSci, 26(2),* 147-179.
2. Aleven, V., Myers, E., Easterday, M., & Ogan, A. (2010). Toward a framework for the analysis and design of educational games. *IEEE DIGITEL*, 69-76.
3. Bjork, R.A. (1999). Assessing our own competence: heuristics and illusions. In Gopher, D., & Koriat, A., *Attention and performance XVII: Cognitive regulation of performance*, 435-459. Cambridge, MA: MIT Press.
4. Black, P., & Wiliam, D. (1998). Assessment and classroom learning. *Assessment in education, 5(1)*, 7-74.
5. Boustedt, J., Eckerdal, A., McCartney, R., Sanders, K., Thomas, L., & Zander C. (2011). Students' perceptions of the differences between formal and informal learning. *ACM ICER*, 61–68.
6. Butler, A.C., & Roediger, H.L. (2008). Feedback enhances the positive effects and reduces the negative effects of multiple-choice testing. *Memory & Cognition, 36(3),* 604-616.
7. Campbell, J., & Mayer, R.E. (2009). Questioning as an instructional method: Does it affect learning from lectures. *Applied Cognitive Psychology, 23,* 747-759.
8. Carpenter, S.K., Pashler, H., & Vul, E. (2006). What types of learning are enhanced by a cued recall test? *Psychonomic Bulletin & Review, 13,* 826-830.
9. Chi, M.T., De Leeuw, N., Chiu, M.H., & LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science*, 18(3), 439-477.
10. Cross, J. (2006). Informal learning: rediscovering the natural pathways that inspire innovation and performance*. San Francisco, CA: Pfeiffer.*
11. Csikszentmihalyi, M. (1990). Flow: The psychology of optical experience*. New York, NY: Harper Perrennial.*
12. Daniel, J. (2012). Making sense of MOOCs: Musings in a maze of myth, paradox and possibility. *JIME*, 3.
13. Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. *ACM SIGCSE Bulletin, 41(1),* 321-325.
14. Garris, R., Ahlers, R., & Driskell, J.E. (2002). Games, motivation, and learning: A research and practice model. *Simulation & Gaming, 4,* 441–467.
15. Gee, J.P. (2003). What video games have to teach us about learning and literacy*. New York, NY: Palgrave Macmillan.*
16. Hays, R.T. (2005). *The effectiveness of instructional games: A literature review and discussion* (Technical Report 2005-004). Naval air warfare ctr. training systems division. Orlando, FL.
17. Johnson, C.I., & Mayer, R.E. (2009). A testing effect with multimedia learning. *J. Edu. Psychology, 101(3),* 621-629.
18. Kapp, K.M. (2012). The gamification of learning and instruction: game-based methods and strategies for training and education*. San Francisco, CA: Pfeiffer.*
19. Karpicke, J.D., & Roediger, H.L. (2007). Expanding retrieval promotes short-term retention, but equally spaced retrieval enhances long-term retention. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 33,* 704-719.
20. Karpicke, J.D., & Roediger, H.L. (2007). Repeated retrieval during learning is the key to long-term retention. *Journal of Memory and Language, 57*, 151-162.
21. Kehoe, J. (1995). Writing multiple-choice test items. *Practical Assessment, Research & Evaluation, 4(9),* retrieved April 2013 from http://pareonline.net/getvn.asp?v=4&n=9
22. Lee, M.J., & Ko, A.J. (2011). Personifying programming tool feedback improves novice programmers' learning. *ACM ICER,* 109-116.
23. Lee, M.J., & Ko, A.J. (2012). Investigating the role of purposeful goals on novices' engagement in a programming game. *IEEE VL/HCC,* 163-166.
24. Malone, T.W. (1981). What Makes Things Fun to Learn? A Study of Intrinsically Motivating Computer Games*. Palo Alto, CA: Xerox.*
25. McDaniel, M.A., Anderson, J.L., Derbish, M.H., & Morrisette, N. (2007). Testing the testing effect in the classroom. *European Journal of Cognitive Psychology, 19(4-5),* 494-513.
26. McNamara, D., Jackson, G., Graesser, A. (2009) Intelligent tutoring and games. *Artificial Intelligence in Education*, 1–10.
27. Mislevy, R.J., Behrens, J.T., Dicerbo, K.E., Frezzo, D.C., & West, P. (2012). Three things game designers need to know about assessment. *Assessment in game-based learning*, 59-81.
28. Pear, J.J. (2004). Enhanced feedback using computer-aided personalized system of instruction. In W. Buskist, V. W. Hevern, B.K. Saville, & T. Zinn, (Eds.), *Essays from excellence in teaching* (Chapter 11).
29. Poehner, M. E. (2007). Beyond the test: L2 dynamic assessment and the transcendence of mediated learning. *The Modern Language Journal*, 91, 323–340.
30. Randel, J.M., Morris, B.A., Wetzel, C.D., & Whitehill, B.V. (1992). The effectiveness of games for educational purposes: A review of recent research. *Simulation & Gaming*, 23(3), 261-276.
31. Riggio, R. E. (2007). Reciprocal peer tutoring: Learning through dyadic teaching. In B. K. Saville, T. E. Zinn, S. A. Meyers, & J. R. Stowell (Eds.), *Essays from excellence in teaching*, (Chapter 10).
32. 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.
33. Sadler, D. R. (1989). Formative assessment and the design of instructional systems. *Instructional Science*, 18(2), 119-144.
34. Shute, V.J. (2011). Stealth assessment in computer-based games to support learning. *Computer games and instruction*, 55(2), 503-524.
35. Shute, V.J., Ventura, M., Bauer, M., & Zapata-Rivera, D. (2009). Melding the power of serious games and embedded assessment to monitor and foster learning. *Serious games: Mechanisms and Effects*, 295-321.
36. Smith, T. (2007). Exams as learning experiences: One nutty idea after another. *Beyond Tests and Quizzes: Creative Assessments in the College Classroom*, 115, 71.
37. Sweller, J. (2006). The worked example effect and human cognition. *Learning and Instruction*, 16(2) 165–169.
38. Vrugt, A.J., Langereis, M.P., & Hoogstraten, J. (1997). Academic self-efficacy and malleability of relevant capabilities as predictors of exam performance. *Journal of Experimental Education*, 66(1), 61-72.
39. Young, J. (2008). "Badges" earned online pose challenge to traditional college diplomas. *Chronicle of Higher Education*.