Gamification to Aid the Learning of Test Coverage Concepts

Eman Sherif Andy Liu Brian Nguyen Sorin Lerner William G. Griswold
{esherif, a6liu, brnguyen}@ucsd.edu {lerner, wgg}@cs.ucsd.edu
Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

***Abstract*—The ability to effectively and efficiently test software is an important practice in software testing that is often under-emphasized in computer science education. Many students find learning about testing to be uninteresting and difficult to learn. This causes numerous students to develop inadequate testing habits, which can be detrimental to their professional careers. To encourage students to develop better testing habits, we used gamification to make the learning experience more engaging and enjoyable. In this paper we explore this idea by integrating gamification and statement coverage into a turn-based game called CoverBot. To test the effectiveness of CoverBot with respect to both teaching statement coverage and increasing engagement and enjoyment, we conducted a user study. We found that gamification makes the learning about statement coverage more engaging and enjoyable while also enhancing the participants performance and understanding of statement coverage.**

***Index Terms*—Computer-aided instruction, gamification, software testing, statement coverage.**

I. Introduction

Today, almost everybody is reliant on software and a minor bug in the code can cause major inconveniences. Examples of software malfunctions include identify theft, data leaks, crashes, or even glitches in video games. Software testing plays an important role in preventing these malfunctions, but requires human work to ensure the production of a high-quality test suite. However, many software engineers struggle to create effective test cases and fail to see the importance of testing. This can result in substandard testing and compromise software quality. An underlying problem is that students in software engineering find testing to be trivial and uninteresting, which can be attributed to a lack of proper motivation during their education or the difficulty of the material [1, 4].

Traditionally, software testing techniques are taught in a classroom where students have little motivation and interest. Moreover, there is minuscule time spent on software testing material compared to other aspects of the software development cycle [11]. In recent years researchers have turned to gamification as a possible solution to increase motivation and engagement within software testing. Gamification is "the application of game-design elements and game principles in non-game contexts" [2, 10].

For example, Microsoft developed a game called Code Hunt that helped students learn how to find and fix bugs and omissions within their code as test cases are added to a test suite [5]. This application was found to be much more efficient and motivating for students to learn coding.

Another successful application of gamification can be seen by a competitive web-based game called Bug Catcher, mainly used for software testing competitions [9]. In Bug Catcher, the game displays a Java class with several bugs in it, and the player must enter a potential test case to detect the bug that fits the requirements.

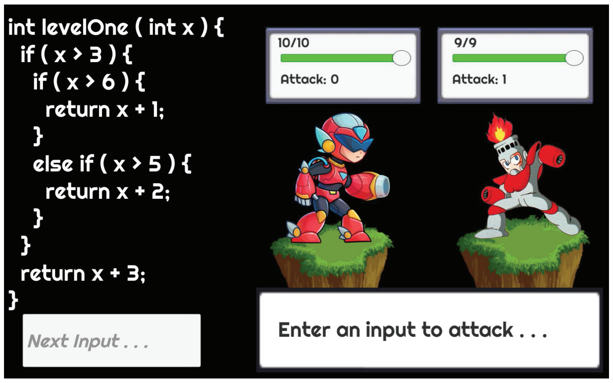
Rojas and Fraser designed the game Code Defenders, which helped players learn how to perform mutation testing. Mutation testing assesses the quality of a test suite by planting artificial faults ("mutants") in a program and measuring how many of them can be found [6]. Code Defenders is a competitive, team-based game where two opposing teams will battle each other within a Java class, utilizing this specific testing technique. One side will be the "attackers" and try to create mutations within the code, while the other side will be creating tests to detect them. Code Defenders utilizes competition, feedback, and incentives as primary gamification features. Their research shows that gamifying software testing can be practical and is a promising field of research to explore.

In this paper we apply gamification to learning the statement coverage concept. Statement coverage is a more elementary testing concept than mutation testing and is more likely to be taught in an introductory programming or software engineering class. An advantage of the statement coverage approach is that there exist automated tools to assess coverage. With statement coverage, the developer tries to create inputs to a code segment such that every statement is executed in order to better verify its functionality. Rather than testing every input the code could execute, statement coverage seeks to both minimize the number of test cases and maximize the amount of code covered [4, 5]. At the same time, statement coverage helps developers detect unused branches, missing statements, and dead code. Statement Coverage is a useful technique for students to learn early in their CS curriculum, which is why we chose to gamify this specific software testing concept.

II. Our Approach

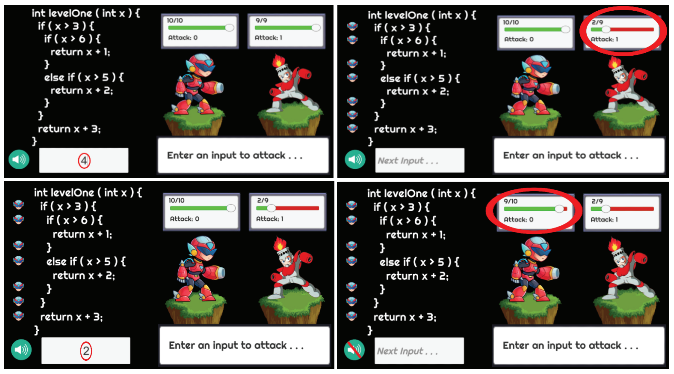
As past studies have shown, games can increase motivation and enhance the learning experience for students learning about a specific topic [7]. There are a wide range of possible game design elements and principles that can be integrated into

978-1-7281-6807-4/20/$31.00 ©2020 IEEE



```
int levelOne ( int x ) {
  if ( x > 3 ) {
    if ( x > 6 ) {
      return x + 1;
    }
    else if ( x > 5 ) {
      return x + 2;
    }
  }
  return x + 3;
}
```

Fig. 1. A screenshot of a level from CoverBot



```
int levelOne ( int x ) {
  if ( x > 3 ) {
    if ( x > 6 ) {
      return x + 1;
    }
    else if ( x > 5 ) {
      return x + 2;
    }
  }
  return x + 3;
}
```

Fig. 2. A walk-through of a level in CoverBot

the process of learning statement coverage. Here we focus on applying gamification features that have been studied to be the most satisfactory for the user [3, 8].

- Graphics/Animations
- Sound Effects/Music
- A Scoring System
- Combat System
- Level Progression

With these features in mind, our goal is to create a game that can facilitate the process of teaching students the importance of statement coverage. We propose a game called *CoverBot* that incorporates gamification with a learning environment for statement coverage. In CoverBot, the player acts as a character who's survivability depends on how effectively the player is able to execute all lines of code in a given level with the fewest amount of inputs as possible. Fig 1 shows an example of a level with the block of code on the left and character models on the right.

The player's goal is to successfully defeat the enemy in the fewest attempts possible (to avoid taking damage) and continue to the next level. Here are some rules to playing CoverBot:

1) The enemy's maximum health value is equal to the total lines of code given.
2) The player types in an input that is passed into the method shown on the left.
3) The damage dealt to the enemy is equal to the number of new lines executed compared to the previous inputs.
4) The enemy will attack if and only if the player doesn't execute any new lines compared to their previous inputs.
5) The enemy's attack value adjusts based on the level that the player is on, increasing as the game goes on.
6) The player's health status carries over to future levels.

Fig. 2 shows a walk through of a level for CoverBot:

1) The upper-left panel in Fig. 2 shows the player typing in the value '4'
2) The top-right panel in Fig. 2 shows that 7 new lines were executed so the player deals 7 damage to the enemy. The lines that were executed have markers placed next to them.
3) The bottom-right panel in Fig. 2 shows the player typing in the value '2'. This input is redundant because it doesn't execute any new lines.
4) The bottom-right panel in Fig. 2 shows the player taking one damage because no new lines of code were executed with the input '2'.

The player continues to go through these steps with different inputs until all lines of code are executed on a given level (or their health goes to zero). If the player is able to execute all lines before the enemy defeats them, they win the level and move onto the next one. However, if the player is defeated they will be prompted to start the level over again. The levels are designed to be progressive, from introductory to more advanced, in order to enable students to build proficiency while sustaining confidence and a sense of accomplishment.

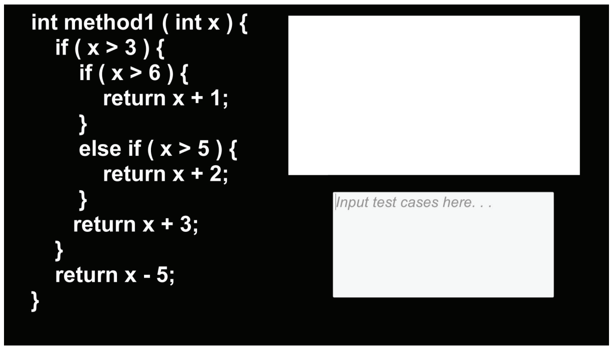
We claim that this game design is able to deliver exercises to students that teach them the benefits of statement coverage, while sustaining engagement by both rewarding them with proper and unique inputs and punishing them if they provide redundant inputs.

III. Experimental Design

To evaluate whether gamification was effective in teaching students about statement coverage, we conducted an experiment using a within-subjects design that focused on the player's preference, engagement, enjoyment, and performance. For the experiment, we have two treatments that each participant went through: a gamified version of CoverBot (experimental group) and a non-gamified version of CoverBot (control group), discussed below. Each version has four levels (or exercises) for the participant to complete. The order of the treatments was randomized and balanced between all participants.

A. Control

To accurately account for the effects of gamification, we created a control version of our game that has the essential gamification elements stripped away (See Section II). In the control version, the player only had the ability to view past inputs and and which lines had been executed by them. A typical level of our control can be found in Fig. 3.

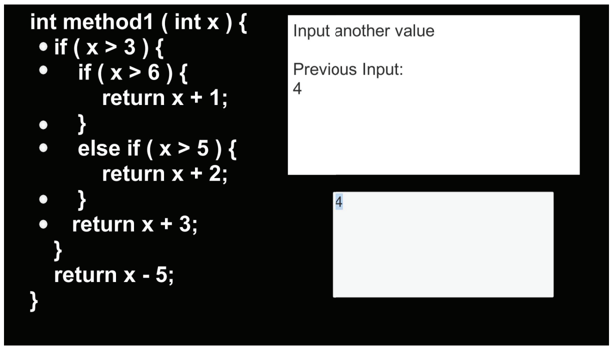


```
int method1 ( int x ) {
   if ( x > 3 ) {
      if ( x > 6 ) {
         return x + 1;
      }
      else if ( x > 5 ) {
         return x + 2;
      }
      return x + 3;
   }
   return x - 5;
}
```

Input test cases here. . .

Fig. 3. A screenshot of a "level" from our control

Notably, we retained feedback on what inputs had been entered, which lines had been executed, as well as a progression of increasing difficult exercises. Thus, the mechanics of how to progress through a "level" (i.e., exercise) in our control remains similar to CoverBot. Fig. 4 illustrates what the user is shown after entering the value '4' on the same level we saw in the previous CoverBot example, Figures 1 and 2.



```
int method1 ( int x ) {
 • if ( x > 3 ) {
 •    if ( x > 6 ) {
         return x + 1;
 •    }
 •    else if ( x > 5 ) {
         return x + 2;
 •    }
 •  return x + 3;
   }
   return x - 5;
}
```

Input another value

Previous Input:
4

4

Fig. 4. A screenshot of a "level" from our control

B. Recruitment and Participants

Participants were recruited using an online interest form. They represent a wide range of software developers, with a majority being college undergraduates studying Computer Science (See Table I). Participants were asked to fill a survey before the experiment began, which provided us with some information about their background.

C. Data Collection

Given the circumstances at the time due to COVID-19, the experiments were conducted over the internet using the Zoom video conferencing app. The participants were asked to download the necessary files and run them while screen sharing so we could observe their play-through of each version.

Each participant was randomly assigned either the gamified or non-gamified version first. We observed and recorded the user's gameplay for each version, focusing on the number of successful inputs. Immediately after each treatment, the participant filled out a Google form survey which asked them to personally rate their levels of engagement, enjoyment, and if it was educational. Also, there was a section for the participant to give general feedback on their experience with that specific version of CoverBot. Once the levels were completed, each participant filled out a final comparison form, which asked them to rate each treatment compared to the other one. After 7 participants went through the study, we noticed that when our participants used the non-gamified program they had significantly more attempts. Therefore we decided to start measuring our future participants' number of successful attempts over the number of total attempts for both the gamified and non-gamified versions. This number would give us the percentage of attempts which were unique and accurate.

IV. Results and Discussion

Through our comparison surveys we were able to gather data on four main points: preference, engagement, enjoyment, and performance. Overall, 90% of participants preferred CoverBot over our control.

Participants were asked to rate their enjoyment and engagement on a 5-point Likert scale. The mean reported enjoyment was 4.35 for CoverBot and 2.95 for the control (See Table II). The mean reported engagement was 4.30 for CoverBot and 3.45 for the control (See Table III). T-tests confirm that the mean enjoyments and engagements are meaningfully different, significant for alpha=0.05 (See Table V).

Participants also performed better while playing CoverBot (See Table IV). For instance, using CoverBot, participants had a ratio of 128 successful inputs to 168 total inputs, a mean accuracy of 76.2%. Participants playing the control, on the other hand, had a ratio of 147 successful inputs to 218 total inputs, a mean accuracy of 67.4%. Moreover, participants using CoverBot were more efficient with their successful inputs; that is, they required fewer successful inputs to complete a level. Participants using CoverBot took a mean of 2.44 attempts to complete a level, whereas in the control it took a mean 3.03 attempts to complete a level. T-tests confirm that the mean accuracies and efficiencies are meaningfully different, significant for alpha=0.05 (See Table V).

TABLE I
Summary of participant demographics (20 total).

Category	Demographic	Number	Proportion
Gender	Female	10	50%
	Male	10	50%
Ethnicity/Race	Asian	11	55%
	Black	3	15%
	Latino	3	15%
	White	2	10%
	Middle Eastern	1	5%
Age	18 - 22	15	75%
	23 - 25	3	15%
	30+	2	10%
Coding experience	$<$ 3 months	2	10%
	$<$ 1 year	3	15%
	$<$ 2 years	8	40%
	$\geq$ 2 years	7	35%
Knowledge of statement coverage	None	12	60%
	Some (varying)	8	40%

TABLE II
PARTICIPANT ENJOYMENT

Version	Mean	Std. Dev.
CoverBot	4.35	0.587
Control	2.95	1.146

TABLE III
PARTICIPANT ENGAGEMENT

Version	Mean	Std. Dev.
CoverBot	4.30	0.571
Control	3.45	0.997

TABLE IV
PARTICIPANT PERFORMANCE

Version	Successful Attempts	Total Attempts	Success Rate	Successful Attempts/Level
CoverBot	128	168	76.2%	2.44
Control	147	218	67.4%	3.03

TABLE V
T-TESTS FOR KEY STATISTICS

Samples	t-value	p-value
Enjoyment	5.016	0.00001
Engagement	6.144	0.00104
Success Rate	2.525	0.00930
Successful Attempts / Level	4.001	0.00026

Additionally, we saw that participants consistently preferred CoverBot, demonstrating that its gamification elements made CoverBot more enjoyable and engaging. Many participants characterized CoverBot as ”fun and enjoying,” other participants described the game as a ”calming and interactive experience”. Also, the added incentive of winning the game made participants perform better. Some participants stated that CoverBot encouraged them to ”input only correct values”. However, when describing the control, many participants expressed how they ”did not care about their inputs”. Furthermore, to gauge their knowledge of statement coverage we asked them to describe it and explain how they would be able to use it when testing their code. 8 out of 12 participants who didn’t know what statement coverage was beforehand were able to accurately describe it by saying phrases similar to ”reaching all lines of the code” and ”trying to execute all lines with few tests cases”. Therefore, CoverBot was also an effective way of teaching statement coverage to our participants.

A. Limitations and Threats to Validity

The study presented here is of a relatively small population, mostly from a single computer science program. Still, this population is highly representative of the target population for CoverBot. The study was also short, essentially a lab study. A longer study embedded in a course could have revealed longer trends regarding engagement, performance, and possibly learning.

B. Future Work

There are additional gamification techniques that might further improve engagement, etc. Also, as just mentioned above, we could run longer experiments with more participants and measure learning outcomes. In the longer term, there are many other ideas to explore:

- *A multiplayer variant of the game*. Interacting with other students might make the material more enjoyable to learn about.
- *A leader board system.* This could instill a form of competition that would keep players engaged and motivated to continue playing the game.
- *A dynamic way of loading in levels.* The levels in CoverBot are hard coded. If we were able to load in levels based on existing methods (code), one could easily increase the diversity and number of levels.
- *Better animations and sounds.* An improvement to the quality of our animations and sound effects could make players feel more immersed in the experience.
- *Incorporation of other programming languages* Changing our game to also include other languages such as C++, Python, or JavaScript can widen the scope of our target audience.

Even more broadly, we could investigate gamification of other aspects of testing, for example test suite reduction, fuzzing, and performance testing.

V. CONCLUSION

In this paper we investigated gamification techniques to aid students in learning statement coverage, an imperative but tedious subject to learn. We designed CoverBot, a turn-based game that incorporates gamification and statement coverage. The principle gamification elements we used in CoverBot include level progression, a scoring/combat system, animation, and sounds.

In order to test the effectiveness of CoverBot we conducted a within-subjects design user study. Through our experiment we were able to measure the engagement, enjoyment, performance, and preference ratings of our participants. We found that CoverBot created a more enjoyable and engaging environment. Additionally, our participants performed better when playing CoverBot and were able to give accurate descriptions of statement coverage.

ACKNOWLEDGMENTS

We thank the Early Research Scholars’ Program, Christine Alvarado, and Vignesh Gokul for this opportunity.

REFERENCES

[1] A. Deak, T. Stålhane, and G. Sindre, “Challenges and strategies for motivating software testing personnel,” *Information and software Technology*, vol. 73, pp. 1–15, 2016.

[2] C. Dichev, D. Dicheva, G. Angelova, and G. Agre, “From gamification to gameful design and gameful experience in learning,” vol. 14, pp. 80–100, 12 2014.

[3] D. L. Kappen and L. E. Nacke, “The kaleidoscope of effective gamification: deconstructing gamification in business applications,” in *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, pp. 119–122, 2013.

[4] E. F. Barbosa, M. A. G. Silva, C. K. D. Corte, and J. C. Maldonado, “Integrated teaching of programming foundations and software testing,” in *2008 38th Annual Frontiers in Education Conference*, 2008.

[5] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. De Halleux, "Code hunt: Experience with coding contests at scale," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 398–407, IEEE, 2015.

[6] J. M. Rojas, T. D. White, B. S. Clegg, and G. Fraser, "Code defenders: Crowdsourcing effective tests and subtle mutants with a mutation testing game," in *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, p. 677–688, IEEE Press, 2017.

[7] K. Welbers, E. A. Konijn, C. Burgers, A. B. de Vaate, A. Eden, and B. C. Brugman, "Gamification as a tool for engaging student learning: A field experiment with a gamified app," *E-Learning and Digital Media*, vol. 16, no. 2, pp. 92–109, 2019.

[8] P. Buckley and E. Doyle, "Gamification and student motivation," *Interactive learning environments*, vol. 24, no. 6, pp. 1162–1175, 2016.

[9] R. Bryce, Q. Mayo, A. Andrews, D. Bokser, M. Burton, C. Day, J. Gonzolez, and T. Noble, "Bug catcher: A system for software testing competitions," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, (New York, NY, USA), p. 513–518, Association for Computing Machinery, 2013.

[10] S. Deterding, R. Khaled, L. Nacke, and D. Dixon, "Gamification: Toward a definition," pp. 12–15, 01 2011.

[11] T. Shepard, M. Lamb, and D. Kelly, "More testing should be taught," *Communications of the ACM*, vol. 44, no. 6, pp. 103–108, 2001.