Teaching Software Testing with the Code Defenders Testing Game: Experiences and Improvements

Gordon Fraser
University of Passau
Passau, Germany

Alessio Gambi
University of Passau
Passau, Germany

José Miguel Rojas
University of Leicester
Leicester, UK

***Abstract*—Code Defenders is a game that aims to engage students with software testing. Players compete over a Java class under test by producing tests and mutants, i.e., artificial faults, scoring points if tests reveal mutants, or mutants survive tests. While initially created as a fun way to introduce students to testing in one-off fun sessions, we have moved to integrating the game as an assessed core component of a software testing course. This has shown great potential in engaging students, but many challenges have been revealed as part of this integration. In this paper we describe how we integrated Code Defenders into a software testing course, and how we improved the game in order to address the resulting challenges.**

***Index Terms*—Software Testing; Gamification; Education**

I. Introduction

Effective software testing is a skill that requires practice and experience. Engaging students with software testing activities so that they gain this experience is challenging, since students tend to prefer spending time on designing and implementing software, rather than testing it. To address this problem, we have developed the Code Defenders game [5], in which students compete over a Java class under test to practice writing effective tests as well as assessing the weaknesses of existing tests.

The Code Defenders game has already been used at many different occasions, and we have evidence that Code Defenders can successfully engage students with writing effective tests and mutants [6]. However, our ultimate objective was to tightly integrate Code Defenders in a software testing course. To this purpose, we made Code Defenders a mandatory part of a software testing course at the University of Passau [1]. This integration into a course with multiple sessions has revealed a number of further challenges that we had to overcome. In particular, repeated sessions with multiple students require not only a smooth user experience for the students, but also for the teacher who has to manage and supervise games. Once the initial excitement of playing a game weakens, students often try to minimise their effort which can impede their learning outcomes. Finally, rewarding students for appropriately engaging with the game is a challenge, since the game score may not reflect actual student performance.

In this paper, we describe our experience of using Code Defenders as part of a mandatory software testing course for undergraduate students at the University of Passau, in which students have to play Code Defenders as part of their coursework. We have refined the integration of Code Defenders into this course over the last three years, adjusting the Code Defenders-related activities to improve the way students interact with the game as well as to align it with the theoretical coursework, and we have added new features that aim to optimise the benefits of using Code Defenders in class.

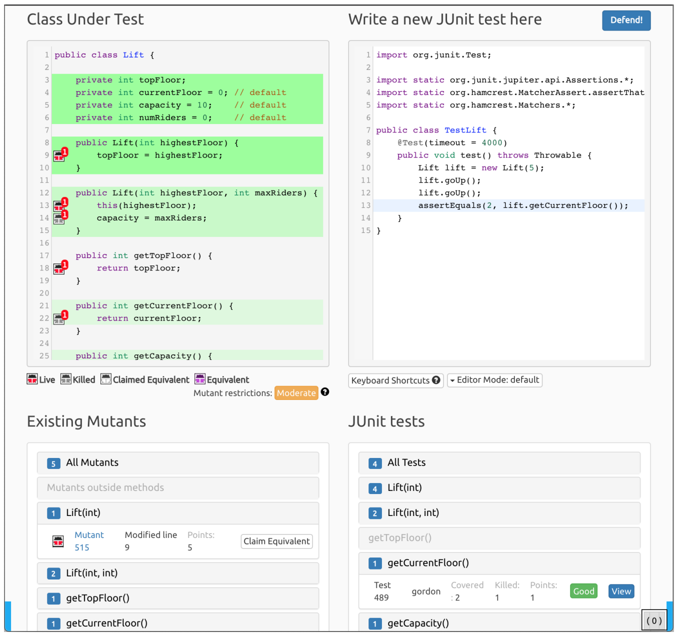
II. Code Defenders

Code Defenders [5] is a competitive game where students play on a Java class under test in one of two roles: attackers and defenders. *Attackers* aim to create subtle mutants of the class under test that evade any existing tests; *defenders* aim to write effective tests that reveal as many as possible mutants. Multiple students can cooperate in teams of attackers and defenders. Teams only get to see partial information about the actions of the opponents: Defenders can see which lines of code were mutated, but they do not get to see the actual changes. Attackers can see which lines have been covered and also how frequently, but they do not get to see the actual tests. This forces players to think about the possible ways in which the class under test can be broken and tested.

The game uses a scoring system in which attackers receive a point for each test their mutant survives; once defenders manage to write a test that detects the mutant they receive as many points as that mutant has accumulated. Equivalent mutants may pose a particular challenge, since defenders cannot decide on equivalence without seeing the underlying source code change. To accommodate for this, defenders can flag mutants as suspected equivalent once they believe to have tested the corresponding code sufficiently. When this happens, the attacker who created the mutant is challenged to an equivalence duel, which requires her/him to write a test that proves the mutant is killable – or to confess that the mutant is truly equivalent, losing all its points. Throughout the game, players can keep track of the points through a dedicated scoreboard, while they can access the overall score of all the players and teams by means of a public leaderboard. Code Defenders is implemented as a web-based game; Figure 1 shows the user interface.

III. Course Integration

After several trials of individual sessions using Code Defenders in various software engineering courses, we decided to make Code Defenders an integral and graded part of a software testing course. In the first instance, this was an optional software testing course [1], open to undergraduate and master level students. The course consisted of weekly two-hour lectures organised around


Class Under Test

```
 1 public class Lift {
 2
 3     private int topFloor;
 4     private int currentFloor = 0; // default
 5     private int capacity = 10;    // default
 6     private int numRiders = 0;    // default
 7
 8     public Lift(int highestFloor) {
 9         topFloor = highestFloor;
10     }
11
12     public Lift(int highestFloor, int maxRiders) {
13         this(highestFloor);
14         capacity = maxRiders;
15     }
16
17     public int getTopFloor() {
18         return topFloor;
19     }
20
21     public int getCurrentFloor() {
22         return currentFloor;
23     }
24
25     public int getCapacity() {
```

Live Killed Claimed Equivalent Equivalent

Mutant restrictions: Moderate

Write a new JUnit test here

```
import org.junit.Test;

import static org.junit.jupiter.api.Assertions.*;
import static org.hamcrest.MatcherAssert.assertThat
import static org.hamcrest.Matchers.*;

public class TestLift {
    @Test(timeout = 4000)
    public void test() throws Throwable {
        Lift lift = new Lift(5);
        lift.goUp();
        lift.goUp();
        assertEquals(2, lift.getCurrentFloor());
    }
}
```

Existing Mutants

5 All Mutants

Mutants outside methods

1 Lift(int)

Mutant	Modified line	Points:	
515	9	5	Claim Equivalent

2 Lift(int, int)

1 getTopFloor()

1 getCurrentFloor()

JUnit tests

4 All Tests

4 Lift(int)

2 Lift(int, int)

getTopFloor()

1 getCurrentFloor()

Test		Covered	Killed:	Points:		
489	gordon	: 2	1	1	Good	View

1 getCapacity()

(0)

Fig. 1: The defender view, showing the source code with mutant locations, an editor for JUnit tests, a table of mutants, and a table of tests.

the standardised International Software Testing Qualification Board (ISTQB) foundation level tester curriculum [3], plus weekly exercise sessions with exercise sheets introducing the technologies implementing the theoretical concepts, including JUnit. We extended this standard course design with weekly two-academic-hour practical session, throughout the whole semester [2]; in these practical sessions students played games of Code Defenders. The practical sessions consisted of each student participating in one game per week, alternating the roles of attacker and defender every week. The difficulty of the classes under test was increased throughout the semester, but we used two similar Java classes in successive weeks to ensure that the students experienced similar challenges when playing as attackers and defenders. In total, each student participated in 12 sessions of Code Defenders, and we evaluated the quality of their mutants and tests throughout the semester to derive a grade for this aspect of the course, in addition the regular exam-based course mark. To ensure that successive weeks allow for increasing difficulty, we have over time added features to Code Defenders such as support for Mockito, Java classes that have additional dependencies (i.e., further classes that are required for testing), or extended assertions (e.g., Hamcrest).

Analysis of our initial course has provided a number of interesting insights (see the report [1] for full details):

- On average, each student submitted 120.54 tests and 158.66 mutants over all games they participated in (average of 19.11 test per game per student, and 35.92 mutants per game per student; maximum of 91 tests in a single game, and 157 mutants in a single game).
- By analysing the branch coverage and mutation scores of test suites produced in the games throughout the course, we observed a clear improvement throughout the semester (Pearson's correlation between session number and coverage is 0.51, p-value < 0.001; for mutation scores the correlation is 0.47, p-value < 0.001).
- Comparison of player actions with exam performance shows a moderate correlation of -0.58 and -0.38 for defenders and attackers, respectively (p-values < 0.001), confirming that better students are more active players.
- We surveyed participating students, and overwhelmingly students like the integration of Code Defenders.

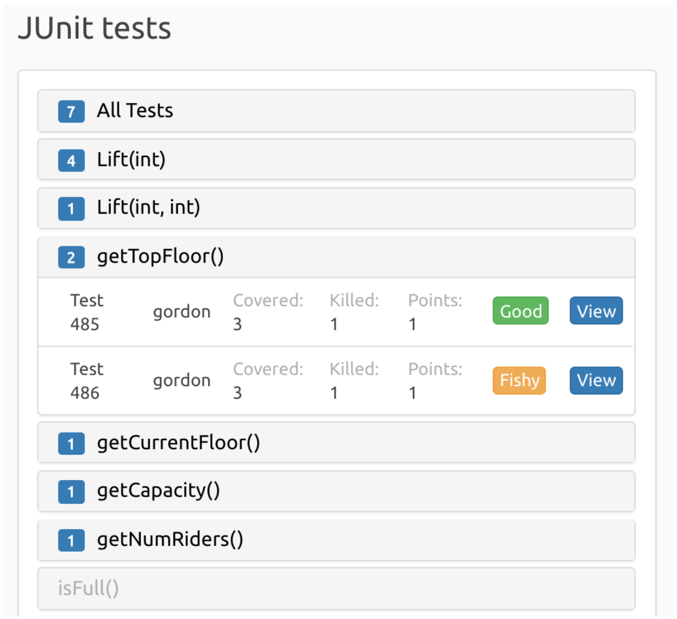
Following the initial instance in 2017/2018, the course was changed to a *mandatory* undergraduate course, but with fewer overall credits. We therefore changed the course assessment to be purely based on coursework, consisting of two parts: One part consists of building test analysis infrastructure (e.g., coverage analysis), and the other part consists of playing Code Defenders. For this we used one introductory session, followed by 6 weeks of Code Defenders, such that each student participates three times as defender, and three times as attacker in Code Defenders games. This reduction of the number of sessions is proportionate to the reduction in overall credits, but we found it to be a good balance between providing sufficient testing experience while maintaining engagement of students in playing the game. The mode of the Code Defenders sessions played as part of the course has not changed from the initial integration [1]. However, we have changed Code Defenders itself to improve this integration.

IV. Improvements to Retain Student Engagement

One problem we observed with students participating in multiple game sessions is that over time some of them tend to minimise their effort rather than engaging fully with the learning objectives. To counter this problem, we added a number of features to the game. These include some general improvements to the user experience (e.g., auto-completion for the code editor, or the possibility to export games to a format suitable for general IDEs), but also some features specifically targeting different player roles.

To further engage defenders, we added the following features:

- Once students are more experienced in writing JUnit tests, they tend to submit more tests to the game. The user interface of Code Defenders proved to be problematic once the number of tests became too large, as it became challenging to navigate amongst existing tests. To counter this problem, we have now organised tests in terms of the methods they cover, allowing players to search specifically for tests of interest (see Figure 2).
- In order to encourage players to not only write tests that are effective, but also *nice* and *maintainable*, we analyse the tests for *test smells* [7] (using the test smell detector developed by Peruma et al. [4]) and indicate them in the test overview (see Figure 2).
- To avoid that defenders tag mutants as equivalent without having attempted to test for them properly, equivalence duels can now only be triggered when the mutated code is already covered by at least one test.



JUnit tests

7 All Tests

4 Lift(int)

1 Lift(int, int)

2 getTopFloor()

Test 485	gordon	Covered: 3	Killed: 1	Points: 1	Good	View
Test 486	gordon	Covered: 3	Killed: 1	Points: 1	Fishy	View

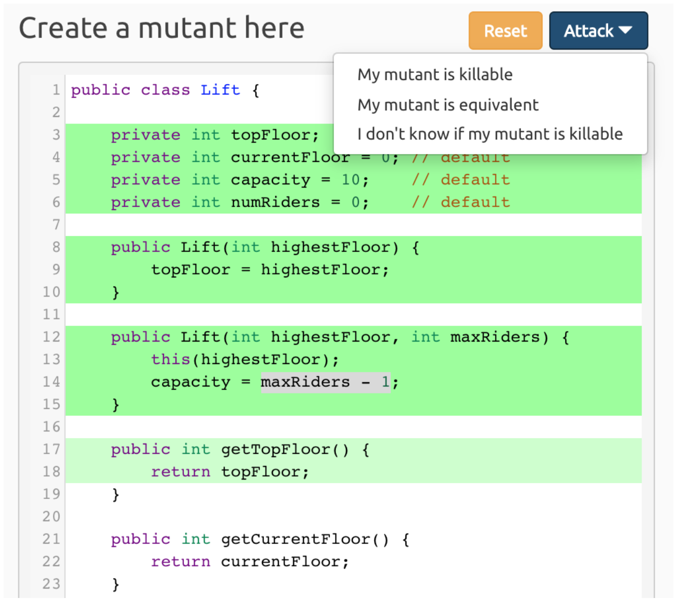
1 getCurrentFloor()

1 getCapacity()

1 getNumRiders()

isFull()

Fig. 2: The test accordion summarises all tests in the game and provides an easy way to navigate them. Tests are shown in terms of the methods of the class under test they cover. In addition, general statistics and smells are shown for each test.



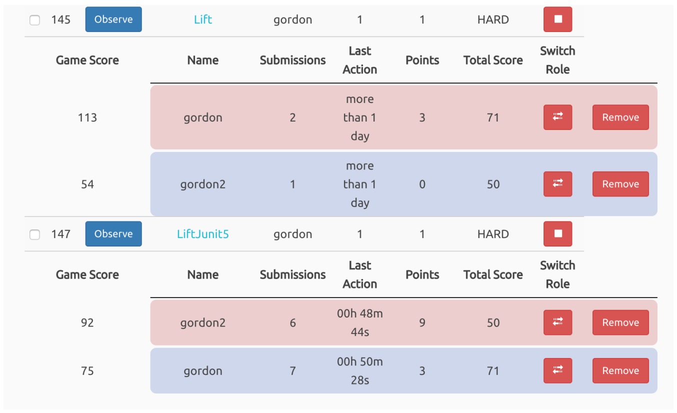
Create a mutant here

```java
public class Lift {

    private int topFloor;
    private int currentFloor = 0; // default
    private int capacity = 10;    // default
    private int numRiders = 0;    // default

    public Lift(int highestFloor) {
        topFloor = highestFloor;
    }

    public Lift(int highestFloor, int maxRiders) {
        this(highestFloor);
        capacity = maxRiders - 1;
    }

    public int getTopFloor() {
        return topFloor;
    }

    public int getCurrentFloor() {
        return currentFloor;
    }
```

Fig. 3: Students can be forced to specify whether they *intend* to produce an equivalent or non-equivalent mutant, encouraging them to think rather than arbitrarily jumbling code. If they are unsure, they can specify that they do not know. Whether or not the classification is correct currently has no effects.

For attackers the challenge of players not properly engaging is somewhat more severe, since it is possible to create mutants by arbitrarily modifying source code without thinking about the existing tests. To better engage attackers with testing activities, we added a number of new features:

- We observed that defenders tend to wait long before triggering equivalence duels. While this is generally desirable for the learning outcomes of the defenders, it potentially means that attackers do not get sufficient chances to improve their testing skills. We therefore added a feature that automatically triggers equivalence duels once a mutant is covered but not revealed by a configurable number of tests. For example, mutants in constructors tend to be executed by every test, and very quickly accumulate points. By automatically flagging them as equivalent, attackers need to think about how to test for their mutants earlier, and this also has the positive side-effect of avoiding that scores are skewed towards attackers.
- To avoid that attackers arbitrarily edit code without thinking about the consequences, we added support to capture their intentions (see Figure 3). For each mutant attackers submit, they have to specify whether they are intentionally submitting an equivalent mutant or a mutant that they believe to be non-equivalent. They can also submit a mutant and specify that they do not know whether it is equivalent or not; however, the aim of this feature is mainly to force them to think about this aspect, so whether or not they select the correct label for their mutant has no effect on the game.
- Defenders tend to require a couple of minutes before they have properly comprehended the class under test and how to test it. During this time, attackers usually create mutants blindly, without knowing where defenders are putting their testing effort. To avoid this problem, we have added support to allow games to be initialised with pre-existing tests, such that attackers immediately have to think about how their mutants could evade existing tests.

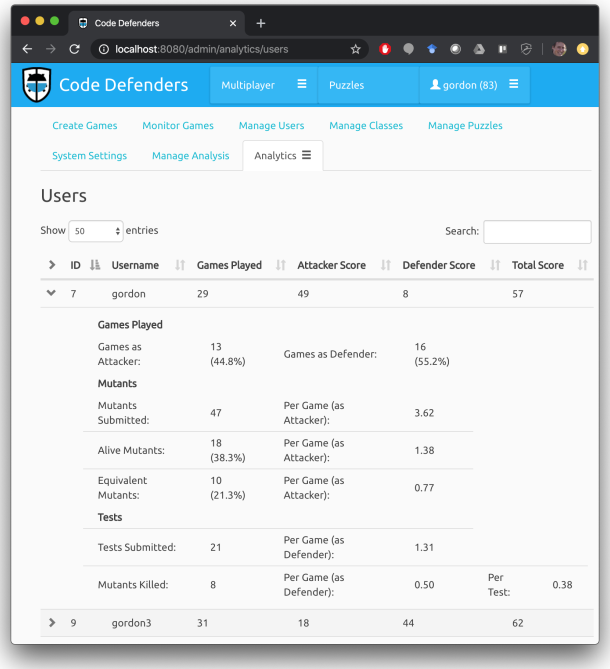


☐ 145	Observe	Lift	gordon	1	1	HARD	■	
Game Score	Name	Submissions	Last Action	Points	Total Score	Switch Role		
113	gordon	2	more than 1 day	3	71	⇄	Remove	
54	gordon2	1	more than 1 day	0	50	⇄	Remove	
☐ 147	Observe	LiftJunit5	gordon	1	1	HARD	■	
Game Score	Name	Submissions	Last Action	Points	Total Score	Switch Role		
92	gordon2	6	00h 48m 44s	9	50	⇄	Remove	
75	gordon	7	00h 50m 28s	3	71	⇄	Remove	

Fig. 4: The administrative interface of Code Defenders allows instructors to create and monitor batches of games. When monitoring active games, an important aspect is the ability to spot students who are either not engaged or are struggling.

V. Improvements to Support Instructors

Managing games as an instructor can be challenging for a number of reasons: Students may arrive late or miss their sessions; the difficulty of the games needs to match the increasing ability of the students to keep them engaged, otherwise students may be disengaged from a game or struggle,



Users

ID	Username	Games Played	Attacker Score	Defender Score	Total Score
7	gordon	29	49	8	57
9	gordon3	31	18	44	62

Games Played					
Games as Attacker:	13 (44.8%)	Games as Defender:	16 (55.2%)		
Mutants					
Mutants Submitted:	47	Per Game (as Attacker):	3.62		
Alive Mutants:	18 (38.3%)	Per Game (as Attacker):	1.38		
Equivalent Mutants:	10 (21.3%)	Per Game (as Attacker):	0.77		
Tests					
Tests Submitted:	21	Per Game (as Defender):	1.31		
Mutants Killed:	8	Per Game (as Defender):	0.50	Per Test:	0.38

Fig. 5: Instructors can view or download statistics on individual players and the Java classes used for the games.

thus inhibiting the gameplay for their teammates and opponents; finally, the participation of the students should be rewarded appropriately by grading their efforts. In order to achieve this, Code Defenders now has a number of features that support the management of games as well as that of student cohorts.

In order to manage games, Code Defenders has an administrative interface in which batches of games can be created. This includes aspects such as alternating the students' roles in successive games and creating teams with balanced ability. Once the games are in action, the administrative interface offers an overview that shows the status of each game, as well as the activities of the individual players (see Figure 4). It has proven particularly helpful to monitor the time since the last action for each student, as typically the best sign for a struggling or disengaged student is the absence of game actions.

To assess the course progress as well as the individual students' engagement, Code Defenders offers a number of statistics and analytics. Figure 5 shows an overview with statistics for individual players, similar statistics are available to monitor the suitability of the classes under test. For grading students, Code Defenders produces *killmaps*, i.e., it executes all tests against all mutants, and then marks mutants and tests as either meaningful or not (i.e., a mutant should survive at least a test but be killable, a test should detect at least one mutant). The use of killmaps ensures that the performance of students is not assessed based on the individual games, since the outcome of a game is heavily dependent on the participating players and not just the player's skills.

VI. Conclusions

After using Code Defenders as a mandatory component of a software testing course for three years in a row, we have overcome a number of challenges and improved Code Defenders. At this point, Code Defenders is stable and easy to use. Our current development efforts are on improving a single-player mode, where players have to solve individual puzzles [6] on their own. We are also working on new and improved game modes to continue improving the player experience.

Code Defenders is open-source and available on GitHub:

https://github.com/CodeDefenders

A public installation is also available to play online at:

https://code-defenders.org

Acknowledgements

This research has been supported by the European Commission through Erasmus+ project IMPRESS 2017-1-NL01-KA203-035259 and DFG grant FR 2955/2-1.

References

[1] G. Fraser, A. Gambi, M. Kreis, and J. M. Rojas, "Gamifying a software testing course with code defenders," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE'2019)*, ACM, 2019, pp. 571–577.

[2] G. Fraser, A. Gambi, and J. M. Rojas, "A preliminary report on gamifying a software testing course with the code defenders testing game," in *Proceedings of the 3rd European Conference of Software Engineering Education*, ACM, 2018, pp. 50–54.

[3] International Software Testing Qualification Board (ISTQB), *Certified tester foundation level syllabus*, https://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html4, 2011.

[4] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba, "On the distribution of test smells in open source android applications: An exploratory study," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, IBM Corp., 2019, pp. 193–202.

[5] J. M. Rojas and G. Fraser, "Code Defenders: A Mutation Testing Game," in *The 11th International Workshop on Mutation Analysis*, IEEE, 2016, pp. 162–167.

[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*, IEEE Press, 2017, pp. 677–688.

[7] A. Van Deursen, L. Moonen, A. Van Den Bergh, and G. Kok, "Refactoring test code," in *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, 2001, pp. 92–95.