CRSG: A Serious Game for Teaching Code Review

Kaan Ünlü
kaan.unlu@ug.bilkent.edu.tr
Bilkent University
Ankara, Turkey

Barış Ardıç
baris.ardic@bilkent.edu.tr
Bilkent University
Ankara, Turkey

Eray Tüzün
eraytuzun@cs.bilkent.edu.tr
Bilkent University
Ankara, Turkey

ABSTRACT

The application of code review in a development environment is essential, but this skill is not taught very often in an educational context despite its wide usage. To streamline the teaching process of code review, we propose a browser based "Code Review Serious Game" (CRSG) with high accessibility, progressive level difficulty and an evolvable foundation for prospective improvements or changes. The application is built as a serious game to reinforce the learning experience of its users by immersing them in its story and theme, helping them learn while having fun. The effectiveness of the game components are measured with a case study of 132 students of 2 software engineering courses. The promising result of this case study suggests CRSG can indeed be used effectively to teach code review. The demo video for the game can be accessed at https://youtu.be/FLnr3p4bhOg, and CRSG itself at: https://github.com/barisardic/crsg.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Applied computing** → **Interactive learning environments**; *E-learning*.

KEYWORDS

code review, software engineering education, serious games, teaching code review

ACM Reference Format:
Kaan Ünlü, Barış Ardıç, and Eray Tüzün. 2020. CRSG: A Serious Game for Teaching Code Review. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3368089.3417932

1 INTRODUCTION

The usefulness of proper code review is undeniable for increasing code quality since it is centered around checking for code defects [15]. However, despite its merit and popularity in the industry [13], it is hard to streamline this process since it has no definitive guideline to adhere to (eg. what to look for while reviewing code may change even among different organizations). Also, the teaching of code review in Computer Science curricula is not wide-spread [9]. As such, it is quite important to find a way for teaching code review in an accessible, streamlined and possibly customizable manner that can be followed easily to work in a number of different scenarios. The following are the scenarios where the teaching of code review is important:

- Individual learning (e.g independent developers)
- Professional learning (such as during orientation sessions for new employees)
- Education (At schools, as early as coding is introduced)

We extended a browser based "Code Review Serious Game" (CSRG) by Ardıç et al. [2]. Current version aims to teach its players the concept of code review along with code review ethic and code examples in the form of playable game levels to create working experience on the subject while having fun as a way to address this problem of teaching code review. Such a tool would potentially have a user-base beyond what is mentioned above, eg. catering to researchers who wish to conduct experiments about code review. Another advantage of it would be its potential for extensibility; the base game can be modified to fit the standards of a certain company, or be specialized for certain programming languages over others.

2 GAME DESIGN

This section consists of 3 main pieces. Subsection 2.1, Game Flow, is a short description of the planned player experience, and the main gameplay loop behind it. Subsection 2.2, Design Highlights and Components, is a list of the more vital elements in the game (such as the game levels), how they are structured, and why so. Subsection 2.3, Implementation Approach, deals with why certain front and back-end methods have been used during the implementation of the game.

2.1 Game Flow

The game has a tutorial, a practice level and 4 actual game levels. The tutorial has a number of code segment examples where each has a single defect in it, together with its classification, description, reason, and how the author could fix it. The practice level is like an interactive manual that teaches the mechanics of the game to the player. The game levels themselves are larger code segments with numerous code defects planted in them. The player has to find all of them, along with their classifications, to be able to complete the levels fully.

2.2 Design Highlights and Components

This section explains the important components in the application which either contribute to the teaching process or improve the user experience while playing.

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3417932

Story and Emotional Palette. Creative director for the video game Journey, Jenova Chen, during his 2013 GDC [4] talk mentions the presence and importance of emotional palettes in creating a product, and especially a game: An emotional palette refers to what kind of audience a game attracts by giving what kind of emotions to them. This could be a sense of achievement, empowerment, socialization, relaxation etc.

Following Jenova Chen's principles, an emotional palette is established for the game. As learning code review (CR) without much stress and having fun while learning it is the primary objective of the game, a theme of exploration felt fitting for it, and the serious game is built around this concept.

To achieve this, Star Trek [8] was taken as an example. In most episodes of the classic Star Trek series, some problem would occur, but through science, bravery and kindness the crew would be able to beat any hardships that came across their way, and learn more about themselves and the galaxy that surrounded them. Remembering this, with the famous catchphrase of the series: "Space: the final frontier. These are the voyages of the starship Enterprise. Its five-year mission: to explore strange new worlds. To seek out new life and new civilizations. To boldly go where no one has gone before!", it was decided that the exploration theme would take a turn for "space exploration", and a protagonist that would have to help fix their way through their problems using the subject of the game, CR, to explore space, where no one had gone before.

Stella. Stella is an effort at the gamification of this serious game beyond visual and auditory effects, to pull the players into the designed theme with interactivity to possibly help them experience the content in a flow state as per Csikszentmihalyi's definition [7], a personality the player can interact with. This personality can be used by the player as if navigating a dialogue screen often seen in popular Role Playing Games (such as Pokémon [6], or Fire Emblem [5]) to both learn tidbits about code review ethic and also to get tips about the mechanics of the game and a narration of its story content.

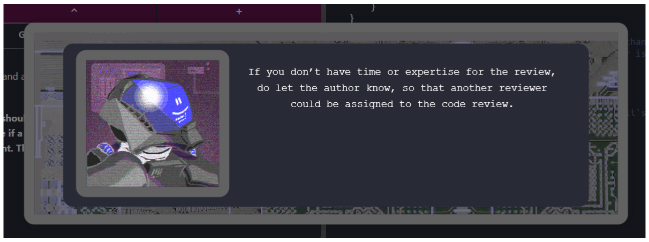

If you don't have time or expertise for the review, do let the author know, so that another reviewer could be assigned to the code review.

Figure 1: A game level with the Stella modal zoomed onto.

This dialogue panel is made so that it automatically appears when the main menu/level map is opened for the first time or a game level is opened at any time. The panel (officially called a modal) can be closed/disabled at-will by clicking outside of it. Clicking on the modal instead will advance Stella's dialogue. At any time during the game in any level or in the level map, the player can click the "log" button on the screen to re-access the modal in case they have accidentally missed it, or wish to re-read some or all of it.

In the narration of the game, Stella calls this dialogue modal her "log", quite similar to the captain's log seen in series such as Star Trek, and uses it to describe her journey, and explain the notes she has taken on CR in the form of CR ethic/group-work advice.

In its creation, attention is given to ease of understanding and use: To keep the UI decluttered, there's no exit button for Stella's log, but the cursor changes shape to a "?" inside the modal, and a pointing hand outside of it. The use of these two nonstandard cursors suggest that clicking in and outside have different functions (as described earlier). The modals include the portrait of Stella in a few variations, to give the log some character in addition to its functionality; it shakes and stutters as if viewed from a cheap camera/screen. One other visual feature of the log is the text in it. It is made in a way that going to the next entry will make the modal write with a typewriter animation (so, flowing letter by letter quickly) and give out a keyboard sound, to give the feel that this is Stella writing in her log. It is expected for this addition to be effective in pulling the player into the game's theme.

The CR ethic tips that come from Stella are a blend of the tips that come from Thoughtbot's Guides [12] and our experiences.

Progress Feedback. In explaining how crowd-sourcing in the form of a game allowed the 2009 "floating duck island" fiasco to be fully uncovered in a very short time, Jane McGonigal draws attention to mankind's attraction to reactivity: "The game interface made it easy to take action and see your impact right away. When you examined a document, you had a panel of bright, shiny buttons to press depending on what you'd found." [14]. Keeping this in mind, it would make sense to make intended player interactions more visually appealing and convenient to use than the rest of the screen and keep a record of the player's achievements, such as their scores in a leaderboard, to better keep them invested with the game.

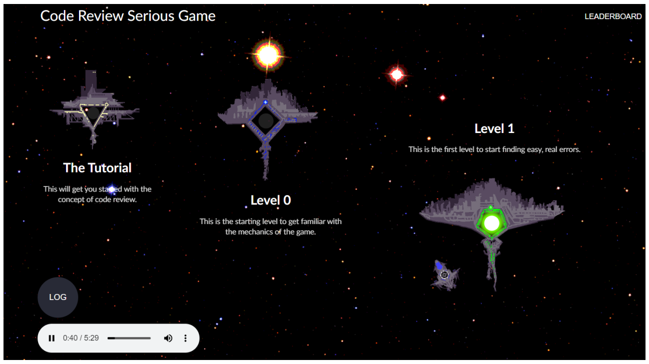
Visual responsiveness is aimed for in the creation of CRSG. CSS/HTML/JavaScript, which is used to construct the game, limits some of this responsiveness, largely due to its instability in regard to zoom levels across different computers and browsers. However, the game still has a number of responsive elements that provide the player with feedback about their progress in the game: The player can ask for hints in any level, and a pop-up will come up from the bottom of the page telling them a single hint specific to that level; clicking the hint button again will cause a different hint to pop up. When the player submits their answers in a level, their answers are lit up in green (the answer is correct), yellow (the correct lines are marked but the stated defect type is incorrect) and red (the answer is incorrect).

UI Components. To further emphasize the emotional palette of the game, the UI of the game is made so that it looks like the player controlling Stella's journey through space. Utmost attention is given to the construction of these UI components to be neither barren, nor overwhelming, providing enough information clearly while not being too busy [1].

Refer to Figure 2 as the components are explained below.

Level numbers and descriptions do not pop-up as tooltips that can be missed, but are instead given under or over space stations in white font without any background. This way, individual levels and what they mean are instantly revealed and are easy to read, but the text does not detract from the appearance of the map, nor does the UI feel too cluttered.

Each space station has a gap in their middle where the button for the corresponding level resides. This space is made part of the



This will get you started with the concept of code review.

This is the starting level to get familiar with the mechanics of the game.

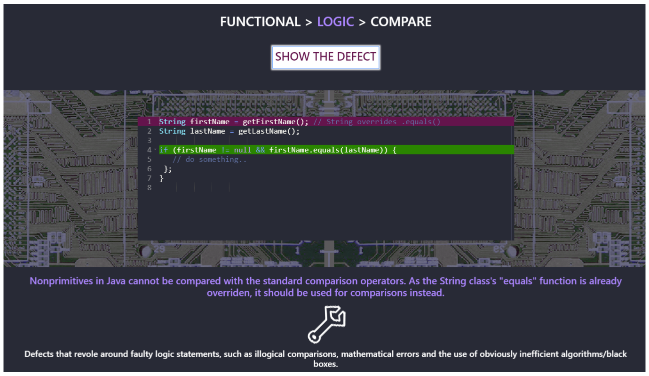
This is the first level to start finding easy, real errors.

Figure 2: The level selection screen of the game.

theme by making them travel relays: Relays Stella has to help repair using CR to be able to travel to the next station/level. To indicate the hardness of the level the player wishes to go into, each successive relay is shaped like a polygon of increasing number of vertices (The tutorial has 3 vertices, the practice level has 4, 1st level has 5 etc). Also, the mouse cursor is changed into Stella’s spaceship, the Crane, to add further into the atmosphere.

There’s music controller on the bottom of the screen, which auto-plays music to increase immersion, but can be stopped, or its sound level adjusted at-will.

Tutorial. The tutorial is made to ameliorate the steep learning curve of CR that comes from the lack of its formality by teaching the players via definitions and examples with solutions. In its current version, it houses 10 different code segments, each with a distinctly different code defect in it. The player can read the definition on any example to see what the type of the defect is alongside that type’s explanation. To make the learning experience smoother, for those who learn better with shapes, the two main defect classes, evolvability and functional defects, are denoted on the tutorial screen with the shape of a brain and a wrench respectively in addition to the text. For those who learn better with colour, different defect sets have unique colours to them.



```java
String firstName = getFirstName(); // String overrides .equals()
String lastName = getLastName();

if (firstName != null && firstName.equals(lastName)) {
   // do something..
 };
}
```

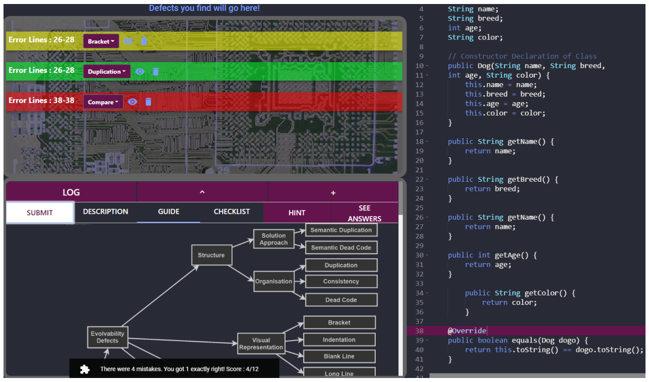
Nonprimitives in Java cannot be compared with the standard comparison operators. As the String class's "equals" function is already overriden, it should be used for comparisons instead.

Defects that revole around faulty logic statements, such as illogical comparisons, mathematical errors and the use of obviously inefficient algorithms/black boxes.

Figure 3: The tutorial of the game after seeing the defect and its explanation in an example.

The player can click the “show the solution” button to see the solution of the shown example defect. When they do, they then see the “see the defect” button. This button reverses the shown solution back to its original way so that the player can compare the defect and the solution at their own pace without having to reload the page and come to the defect they were looking at each time. Also, when the“ show the solution” button is clicked, the defect on that example and how it can be avoided/fixed is explained where the definition of the defect type is given before seeing the solution.

In-Game Helpers. There are four in-game helpers. These are game components aiming to help smooth the learning experience of the player like the tutorial, but without having them leave the level they’re playing, unlike the tutorial. While they are not as descriptive as the tutorial by themselves, they work well in tandem to explain the player some aspects of CR they might not have understood without needing to disrupt the gameplay.



Error Lines : 26-28 Bracket

Error Lines : 26-28 Duplication

Error Lines : 38-38 Compare

There were 4 mistakes. You got 1 exactly right! Score : 4/12

```java
4      String name;
5      String breed;
6      int age;
7      String color;
8
9      // Constructor Declaration of Class
10     public Dog(String name, String breed,
11     int age, String color) {
12         this.name = name;
13         this.breed = breed;
14         this.age = age;
15         this.color = color;
16     }
17
18     public String getName() {
19         return name;
20     }
21
22     public String getBreed() {
23         return breed;
24     }
25
26     public String getName() {
27         return name;
28     }
29
30     public int getAge() {
31         return age;
32     }
33
34         public String getColor() {
35             return color;
36         }
37
38     @Override
39     public boolean equals(Dog dogo) {
40         return this.toString() == dogo.toString();
41     }
42
```

Figure 4: The editor screen, where all levels are played.

Refer to Figure 4 of the editor screen above for the descriptions of the in-game helpers. These 4 helpers are:

Guide. This is one of the tabs on the bottom left of the editor screen. It houses the taxonomy tree of code defects, and concise definitions for all defect types listed below it. These defect types are coloured in the same way as they were in the tutorial to help those that learn with colour better. Also, to help those with reading and concentration problems, each defect type is separated from one another by cells of alternating shades.

Checklist. This is another one of the tabs on the bottom left of the editor screen. It has in it a numbered list of items to check while reviewing code. It is structured in a way that the reviewer is encouraged to review the code starting from more easier to see defects and move on to more contextual, harder to see (and sometimes more situational to the used programming language) ones. It is also structured in alternating cells like the guide to make it more readable.

Show Answers. This is a button on the bottom left part of the editor screen. When clicked, the browser asks in a pop-up whether the player really wishes to see the answers or not to prevent accidental clicks, and if terminated affirmatively, shows the correct defects on the top left part of the editor screen and switches the bottom left tab to the description tab, changing its contents to the explanations of the defects listed above. This way, the player not only sees the correct defects, but also has a chance to read the reason behind them. These explanations are also structured in alternating cells like the guide to make its reading easier on the eyes.

Practice Level. This is an entire level, level 0 to be exact. It describes through comments how the scoring, defect selecting, submitting, seeing answers etc. are done via comments in the editor to the right side of the editor screen. It also encourages the player to try the defect selecting system out by explicitly pointing out parts on the code segment to be selected. This part of the game is integrated to the theme by giving it the flavour of a CR machine's user manual Stella finds in her ship to help her in her journey. Granted that they are following the instructions in this tutorial level, the players have a great opportunity to familiarise themselves with the various helpers on the editor page using this level.

2.3 Implementation Approach

The three main goals this game strives to provide its players/learners are, as per the introduction, a streamlined learning process, high accessibility, and an experience that can be tailored for different standards. CRSG satisfies these constraints with its level design, and browser based implementation. Below is how they manage to satisfy these constraints, and how they affect the use of components in the game, where relevant.

Level Design Philosophy. The levels should become harder not at once, but step by step, to allow the player to grow step by step together with it, and not feel overwhelmed. In CRSG, similar to how the checklist is structured, the levels are designed in a way that they become progressively harder as levels progress: The earlier levels are more likely to have more physically obvious defects (such as bracket or indentation mistakes), while the later levels are way more likely to be dominated by defects that require the code and the language it was written (this was Java in our case) in to be understood clearly to be found out and diagnosed (such as algorithm/performance or memory management defects).

With such progression implemented into the game, the players are expected to need only the first half of the guide and checklist in the earlier levels, and the rest of these components in the later levels (conveniently so, since both of these components are ordered roughly with this progression in mind), essentially splitting the reading load they might have into more digestible chunks.

Web Based Development. To satisfy the premise of accessibility, web based development is assumed for this game. This means anyone who wishes to play the game and has the web address for it can get on playing it without nearly any investment on their part (such as payment for a physical copy, hard disk space for installation, steady and high-speed internet etc). This is because the game runs directly on the client side and communicates with Google Firebase [10] cloud service for authentication and data retention.

One downside of this approach is the limits of HTML/ CSS/ JavaScript programming: Since they do not work quite like most high level programming languages that are made for building games, certain dynamic embellishments for the UI and user pipeline shortcuts had to be cut from the final version of the game, pushing the development instead for a more minimalist design.

The use of frameworks have been avoided to make the evolvability higher for developers that wish to edit the game for their organizations.

3 CASE STUDY AND EVALUATION

We have developed CRSG in order to be able to conduct CR related lab sessions on a number of courses which either focus on technical skills regarding programming, or soft skills and perspectives regarding the industry. We conducted video conference based sessions for Object Oriented Software Engineering and Application Lifecycle Management courses consisting of 52 and 80 students respectively. Moreover, we were able to conduct a survey after these sessions where we asked the students to evaluate the usefulness and their enjoyment of the game components on a 5 point Likert scale. Table 1 presents the averages for each component and overall enjoyment.

Stella's lower usefulness score can be explained with her log lacking substance that can be directly used in game (and instead having real-life advice in it). It could possibly be improved by introducing a mini-quiz into her log screen about code review.

Table 1: BY COMPONENT STUDENT EVALUATIONS

Component	Average (1 to 5)	Standard Deviation
Overall Enjoyment	3.64	0.89
Stella	2.43	1.03
Guide	3.52	1.13
Checklist	3.40	1.18
Tutorial	3.65	1.01
Practice Level	3.67	1.10
Answer Explanations	3.71	1.09
Hints	3.17	1.15

4 RELATED WORK

There are 2 major studies in the literature that are concerned with teaching CR practices directly. Anukarna by Atal et al. [3] is a decision making focused web based game which conveys the best practices regarding the CR process by following a scenario which demonstrates the required resources and outcomes of the process. Guimaraes [11] developed a desktop game that presents a CR flow in which the users play the role of the reviewer by finding defects that are planted into a given code piece that is similar to our design. However, the scope of this work is very limited since it misses quality of life features like syntax highlighting or evaluation on a large user base. To the best of our knowledge, neither of these works are open source or available for public use.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced CRSG, a serious game that aims to teach code review related practices, and presented our design by breaking down the game to its components since, our base design is supported with many components that could help users learn on their own. Such auxiliary components helped create a more complete and useful gameplay experience. After the design, we discuss how this game can be utilized as is or adapted while providing results regarding our experiences with students in software engineering courses. We aim to further improve CRSG by introducing two other multiplayer game modes. One of which will incorporate the role of the author into our game flow to better represent the code review workflows and another one to enable users to conduct reviews as a group.

REFERENCES

[1] P. Alonso. 2020. *Why Do We UI Like We UI?* Retrieved June 1, 2020 from https://forums.warframe.com/topic/1164055-why-do-we-ui-like-we-ui

[2] Baris Ardıç, İrem Yurdakul, and Eray Tüzün. 2020 accepted. Creation of a Serious Game For Teaching Code Review: An Experience Report. In *32nd IEEE International Conference on Software Engineering Education & Training*.

[3] Ritika Atal and Ashish Sureka. 2015. Anukarna: A Software Engineering Simulation Game for Teaching Practical Decision Making in Peer Code Review. In *QuASoQ/WAWSE/CMCE@ APSEC*. 63–70.

[4] Jenova Chen. 2016. *Designing Journey*. Youtube. Retrieved June 1, 2020 from https://youtu.be/UGCkVHSvjzM

[5] Nintendo Corporation. 2019. *Fire Emblem™: Three Houses for the Nintendo Switch™ system – Official Site*. Retrieved June 1, 2020 from https://fireemblem.nintendo.com/three-houses/

[6] Nintendo Corporation. 2020. *Parents' Guide to Pokémon*. Retrieved June 1, 2020 from https://www.pokemon.com/us/parents-guide/

[7] M. Csikszentmihalyi. 2009. *Flow: the psychology of optimal experience*. Harper Row.

[8] CBS Entertainment. 2020. *Star Trek*. Retrieved June 1, 2020 from https://intl.startrek.com/

[9] V. Garousi, G. Giray, E. Tuzun, C. Catal, and M. Felderer. 2020. Closing the Gap Between Software Engineering Education and Industrial Needs. *IEEE Software* 37, 2 (2020), 68–77.

[10] Google. 2012. *Firebase*. Retrieved August 30, 2020 from https://firebase.google.com

[11] Joaquim Pedro Ribeiro Guimarães. 2016. *Serious Game for Learning Code Inspection Skills*. Master's thesis. Universidade Do Porto.

[12] Thoughtbot inc. 2019. *Thoughtbot/guides*. Retrieved June 1, 2020 from https://github.com/thoughtbot/guides/tree/master/code-review

[13] Jil Klünder, Regina Hebig, Paolo Tell, Marco Kuhrmann, Joyce Nakatumba-Nabende, Rogardt Heldal, Stephan Krusche, Masud Fazal-Baqaie, Michael Felderer, Marcela Fabiana Genero Bocco, Steffen Küpper, Sherlock A. Licorish, Gustavo Lopez, Fergal McCaffery, Özden Özcan Top, Christian R. Prause, Rafael Prikladnicki, Eray Tüzün, Dietmar Pfahl, Kurt Schneider, and Stephen G. MacDonell. 2019. Catching Up with Method and Process Practice: An Industry-Informed Baseline for Researchers. In *ICSE (SEIP)*, Helen Sharp and Mike Whalen (Eds.). IEEE / ACM, 255–264.

[14] J. McGonigal. 2012. *Reality is Broken: Why Games Make Us Better and How They Can Change the World*. Vintage.

[15] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study At Google. In *ICSE (SEIP)*, Frances Paulisch and Jan Bosch (Eds.). ACM, 181–190.