GAMFLEW: *Serious game to teach white-box testing*

Mateus Silva[1†], Ana C. R. Paiva[2*†], Alexandra Mendes[2†]

[1]Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, s/n, Porto, 4200-465, Portugal.

[1]INESC TEC, Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, s/n, Porto, 4200-465, Portugal.

*Corresponding author(s). E-mail(s): apaiva@fe.up.pt;
Contributing authors: up201906232@up.pt; afmendes@fe.up.pt;
†These authors contributed equally to this work.

Abstract

Software testing plays a fundamental role in software engineering, involving the systematic evaluation of software to identify defects, errors, and vulnerabilities from the early stages of the development process. Education in software testing is essential for students and professionals, as it promotes quality and favours the construction of reliable software solutions. However, motivating students to learn software testing may be a challenge. To overcome this, educators may incorporate some strategies into the teaching and learning process, such as real-world examples, interactive learning, and gamification. Gamification aims to make learning software testing more engaging for students by creating a more enjoyable experience. One approach that has proven effective is to use serious games. This paper presents a novel serious game to teach white-box testing test case design techniques, named *GAMFLEW (GAMe For LEarning White-box testing)*. It describes the design, game mechanics, and its implementation. It also presents a preliminary evaluation experiment with students to assess the usability, learnability, and perceived problems, among other aspects. The results obtained are encouraging.

Keywords: Software testing, Software engineering, Serious games, White-box testing, Computer science education

1 Introduction

Software testing is a crucial activity within software engineering. However, teaching software testing is challenging as it encompasses many theoretical concepts, requires hands-on practice in a properly set-up environment, and is a constantly evolving knowledge area.

With rapid advances in technology and science, a traditional classroom environment may seem uninteresting to students. This calls for novel approaches that increase students' motivation while keeping them engaged in learning new and challenging concepts. Some examples may be the flipped classroom model, using open-source software in programming projects (with its potential real-life impact), incorporating social interactions into a classroom tool or environment, and gamification.

Gamification is based on integrating game-like elements — such as score, rankings, and badge collection — into student activities to increase their engagement. Serious games incorporate these elements into their structure to provide educational value in addition to entertainment while motivating students to learn. Serious games have been applied to teaching software testing presenting positive results in increasing student engagement [1]. However, existing serious games may not have covered all software testing concepts [2–9].

This paper describes the design, mechanics, and implementation of a new serious game, *GAMFLEW (GAMe For LEarning White-box testing)*, that is being developed in the context of the European project ENACTEST[1] — European Innovation Alliance for Testing Education [10, 11]. *GAMFLEW* features novel ideas, including a single-player mode that introduces an entirely new way to define test cases (more precisely, the test input data needed to achieve a code coverage goal), and a process that allows teachers to add more challenges for students to solve. Other innovative features are the ability for students to submit comments that clarify their rationale for solving challenges and incorporating peer review.

This paper is structured as follows: Section 2 describes past research on gamification and existing serious games for teaching software testing. Section 3 describes a novel serious game, *GAMFLEW*, covering learning goals, its mechanics, architecture, and content. Section 4 presents the validation user study performed with students, followed by conclusions and future work in Section 5.

2 Related Work

The following sections cover related work exploring gamification-adjacent approaches and techniques, followed by existing serious games for teaching software testing topics.

2.1 Gamification

Ramasamy et al. [12] studied students' reactions to a Web-Based Repository of Software Testing Tutorials (WReSTT) supporting collaborative learning. WReSTT includes social networking features that encourage student interactions in testing by awarding points. One observation from the study was the frustration of students who

[1] http://enactest-project.eu/

had difficulty designing appropriate test cases when receiving penalties. This happened as students lacked instruction in testing methodologies and techniques, which was resolved using the Software Engineering and Programming Cyberlearning Environment (SEP-CyLE), a component of WReSTT. Students recognized that software testing knowledge positively impacted their programming skills and that SEP-CyLE was an excellent support for acquiring that knowledge, which was visible in the improvement of scores. The authors finally concluded that software testing knowledge allowed students to *quickly move programming practices into working memory on demand* when it was applied gradually.

In 2019, Ferreira Costa and Bezerra Oliveira [13] introduced exploratory testing in teaching. The approach focused on the design and execution of tests in a free, concurrent manner, and incrementally builds on past exploration. The authors defined a syllabus for undergraduate courses in computer science, including some elements of gamification: free lunch (granting a prize for reaching a goal), achievement medals, progress bars, and group quests, among others. In collaboration with experts through peer reviews, an evaluation identified some necessary changes to be implemented. The approach followed work on a gamified framework inspired by a pirate theme, which, when run on an experiment with students, yielded over 70% of success [14].

Elgrably and Oliveira [15] also applied gamified competition mixed with agile testing principles in a software quality course. Students learned about test-driven development and were attributed points in activities associated with case studies. Results report satisfactory to very satisfactory student performance while also unanimously pointing towards a positive outlook on using gamification to address motivation. The use of penalties (for example, for using mobile devices during class) is highlighted as a way to keep students' attention.

Gomes and Lelli [16] created a game-based learning approach called *GAMUT* (GAMe-Based Learning Approach for Teaching Unit Testing), designed around a narrative of monster-fighting in the Middle Ages. The game is inspired by *Mastermind* and is made to be played individually. It introduces the *given-when-then method*, related to unit testing and white-box testing, in a trial-and-error fashion. Students then performed practical activities based on the challenges played in the game. While some issues are reported by the authors, namely that students experience difficulties understanding some concepts, they do believe the approach successfully provided an experience that was not merely theoretical and introduced complex techniques to students. Students reacted very positively to the approach and the dynamic it introduced in classes.

The impact of gamification on software testing education was analysed by de Jesus et al. [17]. Two groups of students were evaluated using quizzes, where one group had access to a gamified environment for their learning. Motivation was higher among students who used gamified techniques. Results were negative regarding performance, but student feedback proved that gamification helped capture the students' attention and appreciation.

2.2 Existing Serious Games

Some serious games have been developed to help in teaching software testing concepts.

Clegg et al. [2] developed *Code Defenders*, which incorporates the main activities of mutation testing (as per the paper: *"creating mutants, killing mutants [and] checking mutant equivalence"*). The game features an attacker and defender mode — simply put, an attacker creates mutants for the defender to try to kill. However, defenders can claim equivalence of a mutant, forcing the attacker to either forfeit or write a test case that kills it and disproves equivalence. The most distinguished feature of the game is its competitive loop; just one pair of players creates a "cat-and-mouse" game experience between players who fight to get the highest score and win the game. The authors propose different strategies to use the game in a testing curriculum and propose ideas to teach concepts, such as statement coverage, branch coverage, and dataflow testing.

In a later preliminary report on the application of *Code Defenders* in a software testing course, Fraser et al. [3] used a classroom environment to see how students reacted. The authors addressed scaling, which allowed for dealing with 120 students, and created an administrative interface. Students of equal ability were grouped, and the authors acted as monitors of the gaming sessions, motivating students to play by participating in the games themselves. Certain measures were taken to prevent players from unfairly manipulating game mechanics. According to the authors' observations, the game succeeded in engaging students at all levels, and students often debated beyond the time limits of sessions. The scores proved very enticing for students, who tried to boost theirs using *"dubious game strategies"*. The authors also point out some potential difficulties for struggling students, which may inform changes in gameplay. Furthermore, finding the right code complexity for both player roles is quite hard, along with keeping student engagement by scaling the complexity and challenge of the game throughout the semester. Overall, the conclusions are very positive.

Another example that is less focused on competition is *TestEG*, by Oliveira et al. [4], which creates a simulated environment of a software testing team as a "Test Manager." Players have a budget to appropriately spend on training their team. The game's most distinguishable features are the possibility for an educator to interact with the game in a controlled manner (by managing player authentication and adding questions that teams can present) and the *RPG*[2] nature of the game, with 3D environments and character selection as exemplary features.

Soska et al. [5] developed an experimental card game with their efforts focused on a framework based on constructivism — which states knowledge grows via a *"process of active construction"*, as per Mascolo [18] — and the theoretical content of the *ISTQB* (International Software Testing Qualifications Board) [19] foundation level. Players used cards based on *Magic: The Gathering*. The game was mostly theoretical and covered concepts and explanations, which could be played solo or competitively, resembling studying flashcards. The gameplay allowed students to share their task solutions and work collaboratively, which was very valuable in the learning process. In the end, students agreed that the game allowed them to gain new insights and findings about software testing content and deepen their knowledge.

Fraser [6] researched gamification in programming education, highlighting the potential for its inclusion in software testing, and analysed several games, including:

[2]Role-playing game.

CodeHunt [20], an online game focused on coding but that is related to testing, as players test their written code by comparing its performance to a test suite; *CodinGame* [21], which is similar to *CodeHunt*, allowing players to check their code against tests; and *CodeDefenders*. The author points out how gamification can be leveraged to better engage students with testing, mentioning examples like *Bug Hunt* [22] (which applies different testing techniques) and the use of storylines and quests by Bell et al. [23]. Another interesting example is that of *CodeFights* [24], whose sole purpose is recruitment through directed courses in different topics (since companies may pay to be in contact with top players)[3], and once again involves tests. Testing games may also be made with a purpose in mind, like crowd-sourced testing of mobile and web apps — for example, Logas et al. [25] gamified the process of summarizing loops in Xylem.

Valle et al. [7] similarly covered educational games and developed their own testing game, named *Testing Game*. The game addressed functional, structural, and defect-based testing. Each of these techniques had a set of different stages, which may vary from combat-based gameplay (*"[...] they must eliminate the enemies and the invalid vectors."*) to gameplay resembling the solving of worksheet exercises (*"[...] students must observe the Def-Use graph of the Bubble Sort program and fill in a table that demonstrates in which nodes of the graph the variables were defined."*). The authors covered the quality and usability of the game with the collaboration of students: 86.6% positively evaluated the motivation gained from playing the game, 87.7% of students enjoyed the user experience, and 82.23% of students positively considered the game's learning factor.

Ribeiro and Paiva [8] presented a serious game for teaching software testing, *iLearnTest*. This game is different from previous games, such as *U-TEST* [26], *Bug Hunt*, and *TestEG*: *U-TEST* only covers black-box and unit testing; *Bug Hunt* is relatively short, being comprised of only five classes[4]; and *TestEG* is a simple quiz game. *iLearnTest*, however, covers ISTQB topics and may be used to help study for this certification while adding gameplay challenges to captivate users to learn: platforming, (*Hangman*-like) concept guessing, concept separation, path-finding and, ultimately, quizzes. When applied in a teaching context, students who used the game to study performed better in a multiple choice questions exam than those who did not[5].

For a different experience, Materazzo et al. [9] developed an app called *Unit Brawl*, inspired by the battle royale genre with only one winner. In the developed app, students are against their colleagues — beyond receiving a point whenever their submissions are admissible by the app, students gain points whenever their written tests (which pass on their code) fail on another student's code and whenever other students' tests pass on their code. The game features a leaderboard and avatars, where accumulated points can be converted into avatar customization items. Competition is named the main drive of the app. A preliminary evaluation of the app showed that it behaved consistently with the authors' expectations. Still, a thorough evaluation within an object-oriented programming course is stated as immediate future work.

[3]When accessing the website provided, we are redirected to "CodeSignal," assumed to be a rebranding of the mentioned game.
[4]Perceived as levels.
[5]Paraphrased from author translation. Pages 1 and 3-6.

Many serious games aim to teach software testing concepts. To our knowledge, only two explicitly name white-box testing in their description [4, 7]. More than once, ISTQB [19] is used or cited as a standard for the content of a serious game [5, 8].

GAMFLEW aims to teach white-box testing concepts, allowing teachers to adapt game content to the needs of their students. It differs from previous work by integrating features for both students (the challenges) and teachers (for designing the challenges). Some features include an interactive way that allows students to build the test cases for overcoming challenges and the possibility for teachers to monitor their students' learning and evaluate them more closely via students' submitted comments.

3 *GAMFLEW*

GAMFLEW is a serious game for teaching white-box test case design techniques, where students define the test input data needed to achieve code coverage objectives defined in challenges. Five different code coverage concepts are featured.

A challenge encompasses a snippet of code to analyze, the code coverage objective to achieve — such as "decision coverage of line N" —, and a *Checkers* board to define the test inputs. The game encourages users to interpret the code snippets in order to pass the challenge with the right combination of board interactions.

To check if a challenge was *passed*, all moves made on the board are verified by the game. This may include piece movements, adding or removing pieces, and piece coordinates. Players have more freedom of movement than in a game of Checkers as, for example, they can move pieces outside the boundaries of the board. The challenges also feature hints to help struggling users.

Players get rewarded with points after submitting an attempt and commenting on it — in a way, to "show their work". Submitting the comment generates a score and logs relevant statistics of their gameplay, shown in real time in the game's interface and accessible by the teachers.

Beyond the two sets of challenges provided for players as base content, special teacher-exclusive features allow teachers to edit and manage the challenge catalog by adding new challenges. Teachers may also play any challenge in the game, or check player submissions and performance if they wish to.

3.1 Description

This Section describes *GAMFLEW* (concepts taught, goals, game elements, rules and teachers-exclusive features) while the following section (3.2) gives details about its implementation.

3.1.1 Code Coverage Concepts

The following white-box testing criteria are featured in *GAMFLEW*:

- *Statement coverage*: the extent to which lines of code are executed by test cases.
- *Decision coverage*: the extent to which decisions (both True and False outcomes) are executed by the test cases.
- *Condition coverage*: the extent to which conditions (both True and False values) are executed by test cases.

- *Condition-decision coverage*: both condition and decision coverage.
- *Modified Condition Decision coverage*: the extent to which test cases exercise Boolean conditions inside decisions independently affecting the decision outcome while covering all possible Boolean values of the conditions.

3.1.2 Objectives

The game is structured around challenges, which are overcome by a variable number of test cases, e.g., statement coverage needs one test case while decision coverage needs two.

A test case is built by interacting with the board, an 8x8 grid. Required interactions may be described as plays from Checkers — moving a piece or making a King, for example —, but there are additional possible movements — e.g., moving pieces out of bounds or stacking them. The game saves the interactions (i.e., piece movements on board), the current, and final board states. In this game, we do not ask for test case expected results, and players do not need to submit the test code. This approach intends to focus students' attention on testing concepts without worrying about the particularities of testing frameworks, such as JUnit.

A challenge has an associated code snippet and a test coverage goal. Players analyze the code and design tests by interacting with the board to achieve the coverage goal. All interactions made by a player are logged, and the saved information is used when evaluating if an attempt passes or fails the challenge's objective.

It is important to highlight that, currently, *GAMFLEW* focuses on teaching the code coverage concepts mentioned above and not on evaluating the effectiveness of tests when detecting bugs. The objective is to define input data that allows the execution of the code in a way that achieves the desired coverage. Also, we do not ask for expected test results, only the test input values. So, here, the test cases may be seen as a product of understanding the code snippet (written in JavaScript), instead of being created to try and reveal bugs (test effectiveness). Indeed, it is only necessary to hit the code coverage objective to pass a challenge — this is the game's main objective.

Challenges may be created by teachers. To better illustrate a challenge, consider the code snippet example in Listing 1. A possible challenge goal for this code would be "Statement coverage of line 4". To achieve this goal, players must move the pieces on board so as the boolean expression in lines 2 and 3 is evaluated to *True*. This may be achieved, for example, by moving a piece from *start* $(4, 2)$ to *destination* $(-1, 2)$.

```
1  function F(board, start, destination) {
2     if (destination.x < 0 || destination.x > 7 ||
3        destination.y < 0 || destination.y > 7) {
4        return false;
5     }
6  }
```

Listing 1: Example function (out of bounds movement).

3.1.3 Board

The board interaction is inspired by *Checkers* but has been adapted and expanded with new possible interactions. Most of these are illegal in the board game, such as moving pieces out of bounds and stacking pieces. Although behaviour like stacking pieces cannot happen in a Checkers game, from the tester's point of view, it makes sense to test whether or not undesired behaviours are possible in a Checkers game implementation. At the same time, this game mechanic shifts the focus to the understanding of code snippets.

3.1.4 Hints

Each challenge has an unlockable hint to help potentially struggling players to pass challenges. These hints are shown whenever a failing attempt to solve a challenge is submitted and are highlighted on the interface when such a condition is met.

This mechanic allows us to prioritize player understanding of the game mechanics and thus guide those who are struggling. The end goal of the hints then is to have all players able to pass all challenges, even if by being helped.

3.1.5 Comments

To successfully submit an attempt, players must also submit a comment that briefly explains their rationale when building their solution. This feature was initially included to encourage proper interaction between teachers and students. The comments can be valuable, as they allow students to express their difficulties to their teacher and may also provide the teacher with a clearer picture of whether their students understand what is being taught.

As an example, a student who submits a poor comment on their submissions — i.e., a key smash or a completely uninterested, overly short sentence — can signal that the student has not given the challenges enough attention or is unmotivated, signaling the need for help from a teacher.

Comments from passing attempts are also used as anonymous hints among students (accessible after submitting any attempt), who can access this extra help to beat a particularly difficult challenge. This feature is akin to the incorporation of peer review between students.

3.1.6 Score

Beating a challenge grants the player points, and each challenge has an attributed score that is aligned to a challenge's difficulty level. These points are used as a technique to grow the players' motivation to play and beat as many challenges as possible. The number of failed attempts has no impact on the earned points — passing a challenge will always grant the same amount of points to a player.

Using penalties to punish students who fail to pass a challenge was considered. However, we decided against this because of our understanding that negative feedback, such as penalties, may be disconcerting to students, as opposed to motivating (as was observed by Ramasamy et al. [12]). As such, there are no penalizations on challenge scores.

3.1.7 Teacher-Exclusive Features

The game includes an exclusive view for teachers, where it is possible to create challenges and to check students submissions and their overall performance. We highlight these features as a way to expand the game's base content, while also allowing it to be tailored to the learning goals the teacher wishes students to meet. By providing the possibility for teachers to expand the game's content, the game may have seemingly infinite content — only limited by the teachers' imagination.

3.2 Implementation

The following subsections detail the implementation of *GAMFLEW*.

3.2.1 Architecture

GAMFLEW was developed as a monolithic app. There are only two components: a backend and a frontend. The backend includes the database (with the base content seeded into it) and the API that allows communication with the frontend.

Access to the API is restricted to authorized requests via a bearer token generated by OAuth2. Without this token, registering a user is the only possibility.

A representation of *GAMFLEW*'s architecture can be seen in Figure 1.

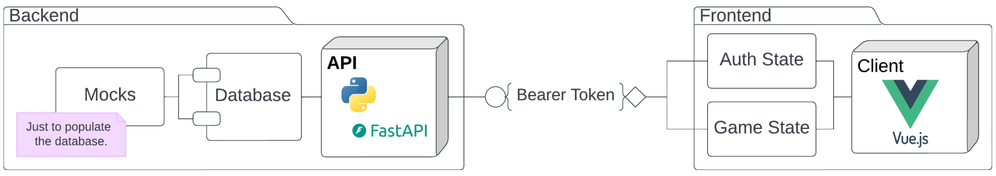


Fig. 1: Architecture diagram.

3.2.2 User Interface

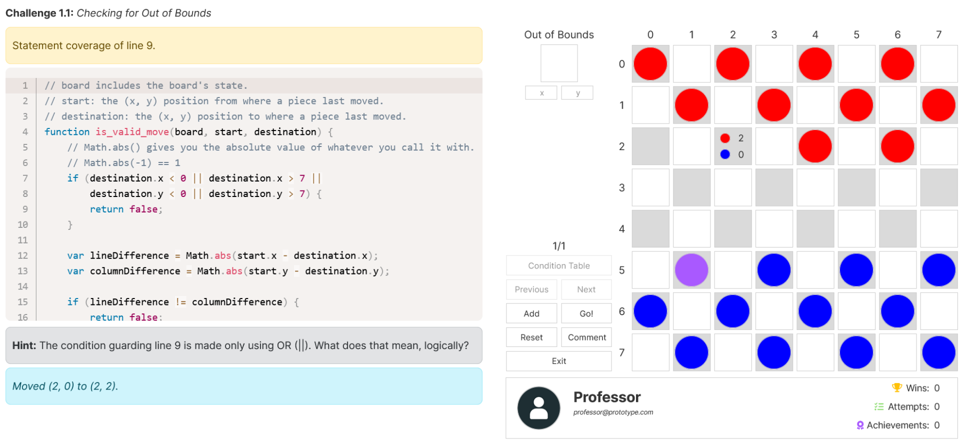
GAMFLEW's appearance is inspired on `boardgames.io`'s implementation of *Connect 4*, which can be (partially) seen in the Wayback Machine[6].

The main gameplay happens on the challenge page, which is divided into two horizontal sections. The left ection lists the challenge's details (title, objective, code snippet, hint, and interaction logs), while the right Section is almost completely occupied by the game board, as shown in Figure 2.

3.2.3 GAMFLEW's Functionalities

GAMFLEW's challenges are saved in JSON format in the database, with most key-value pairs having numerical or textual format. Challenge's expressions are saved textually in JavaScript format. They are then evaluated to be executed as-is — the

[6]The Wayback Machine was used to access an older version of the website. This was because the appearance of the website has changed (by accessing this link, one can see the new appearance).



Challenge 1.1: *Checking for Out of Bounds*

Statement coverage of line 9.

```
1  // board includes the board's state.
2  // start: the (x, y) position from where a piece last moved.
3  // destination: the (x, y) position to where a piece last moved.
4  function is_valid_move(board, start, destination) {
5      // Math.abs() gives you the absolute value of whatever you call it with.
6      // Math.abs(-1) == 1
7      if (destination.x < 0 || destination.x > 7 ||
8          destination.y < 0 || destination.y > 7) {
9          return false;
10     }
11
12     var lineDifference = Math.abs(start.x - destination.x);
13     var columnDifference = Math.abs(start.y - destination.y);
14
15     if (lineDifference != columnDifference) {
16         return false;
```

Hint: The condition guarding line 9 is made only using OR (||). What does that mean, logically?

Moved (2, 0) to (2, 2).

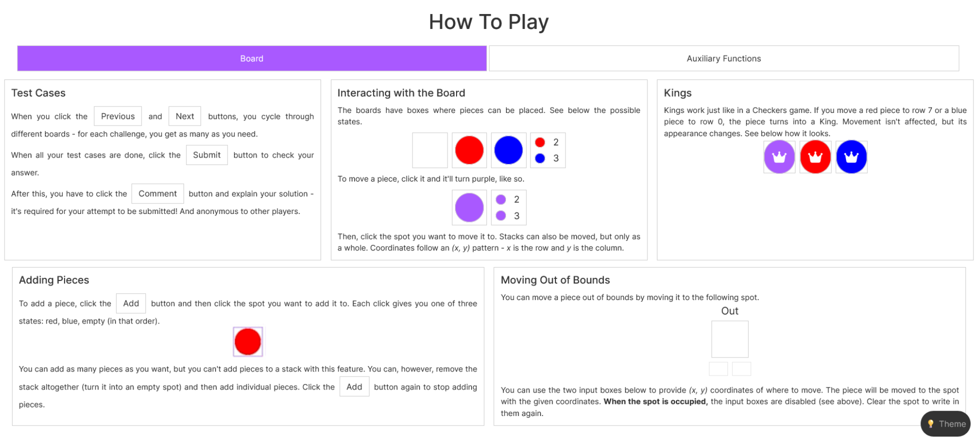
Professor
professor@prototype.com

Wins: 0
Attempts: 0
Achievements: 0

Fig. 2: View of a challenge (Challenge 1.1) and Game Board.

game was programmed to handle this. The challenge's objective is highlighted in yellow, above the code snippet. Hints are highlighted in gray and black, below the code snippet. Attempt comments are submitted in a popup window.

After submitting their comment and successful attempt, users are rewarded points, which are shown live in the game (below the game's board, as seen in Figure 2) and shown in a notification to the user. An "How to Play" page provides instructions and information about the game, as shown in Figure 3.



How To Play

Test Cases

When you click the Previous and Next buttons, you cycle through different boards - for each challenge, you get as many as you need.

When all your test cases are done, click the Submit button to check your answer.

After this, you have to click the Comment button and explain your solution - it's required for your attempt to be submitted! And anonymous to other players.

Interacting with the Board

The boards have boxes where pieces can be placed. See below the possible states.

To move a piece, click it and it'll turn purple, like so.

Then, click the spot you want to move it to. Stacks can also be moved, but only as a whole. Coordinates follow an *(x, y)* pattern - *x* is the row and *y* is the column.

Kings

Kings work just like in a Checkers game. If you move a red piece to row 7 or a blue piece to row 0, the piece turns into a King. Movement isn't affected, but its appearance changes. See below how it looks.

Adding Pieces

To add a piece, click the Add button and then click the spot you want to add it to. Each click gives you one of three states: red, blue, empty (in that order).

You can add as many pieces as you want, but you can't add pieces to a stack with this feature. You can, however, remove the stack altogether (turn it into an empty spot) and then add individual pieces. Click the Add button again to stop adding pieces.

Moving Out of Bounds

You can move a piece out of bounds by moving it to the following spot.

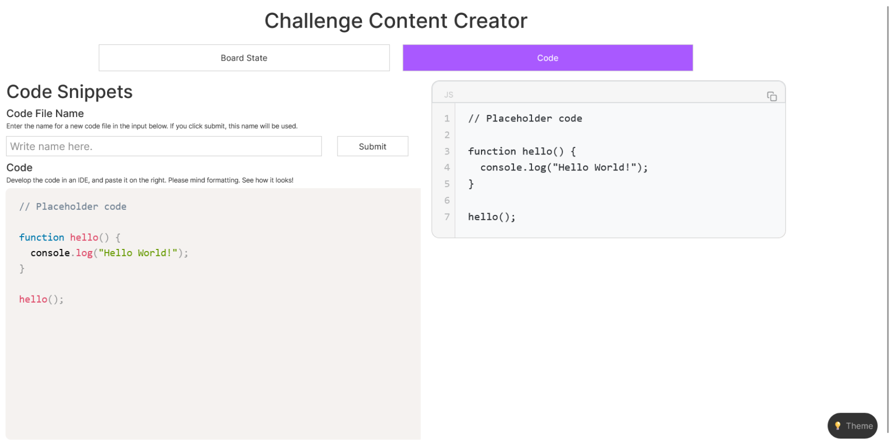
You can use the two input boxes below to provide *(x, y)* coordinates of where to move. The piece will be moved to the spot with the given coordinates. **When the spot is occupied,** the input boxes are disabled (see above). Clear the spot to write in them again.

Fig. 3: *How to Play* page (Instructions tab).

In terms of **board interaction**, the following functionalities are available (see Figure 2):

- **Basic Movement**: if we click on a piece, it turns purple, meaning it is selected. Clicking on any other board position will move the piece to the chosen destination.
- **Stacking**: more than one piece may be moved to the same destination.
- **Out of Bounds Movement**: a piece may be moved to any position outside the 8×8 board grid. The player may choose the destination coordinates, though these default to position $(-1, -1)$.
- **Add Mode**: by clicking the *Add* button on the button list to the left of the board, it is possible to change the number of pieces in the board, whose contours turn purple. Clicking on an empty position adds a red piece to it. Clicking a red piece turns it blue and clicking a blue piece makes the spot empty. Clicking on a stack removes it from the board.
- **Nesting of test cases**: the *Previous* and *Next* buttons allow the player to cycle through test cases, and are only available when a challenge requires more than one test case to be submitted.
- **Test case submission**: the *Go* button submits all test cases made by the player. The user receives feedback right after submitting their comment, either a (red) warning that they failed, allowing for further attempts, or a (green) message of congratulations.
- **Board Resetting**: the *Reset* button reinstates the initial board state of the challenge.

Some functionalities are **teacher-exclusive**. For example, teachers can create new challenges for the game and manage them. A challenge requires defining its code snippet (in-game, these are called "code files"), an initial board state (to initialize the board when the challenge is opened — the same initial state is applied to all test cases whenever many are needed), and the evaluation expressions.



Enter the name for a new code file in the input below. If you click submit, this name will be used.

Develop the code in an IDE, and paste it on the right. Please mind formatting. See how it looks!

```
// Placeholder code

function hello() {
  console.log("Hello World!");
}

hello();
```

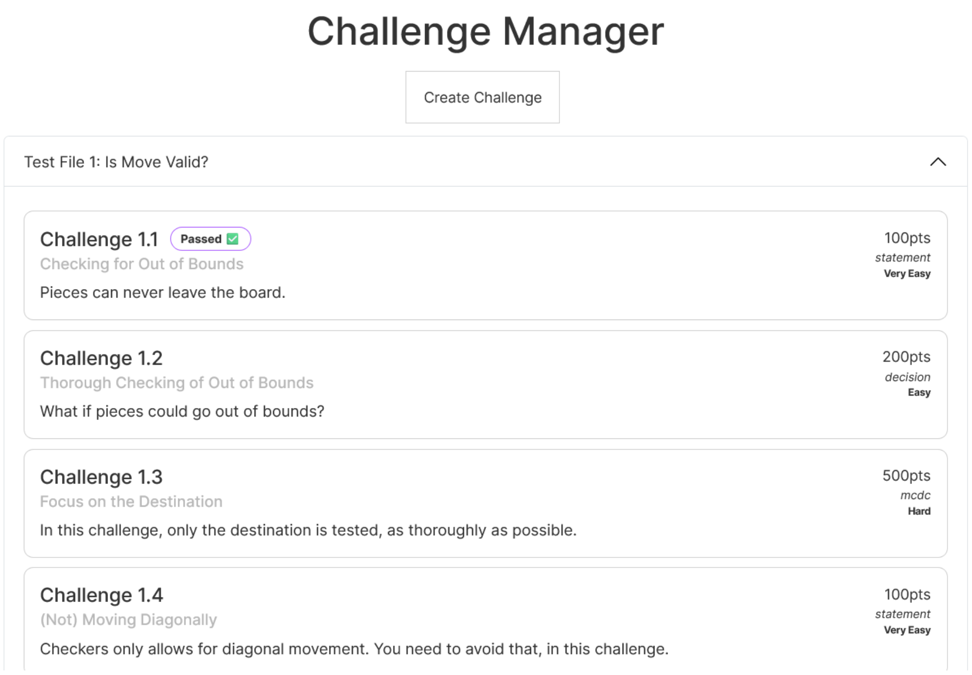
```
// Placeholder code

function hello() {
  console.log("Hello World!");
}

hello();
```

Fig. 4: *Challenge content creator* page (Code tab).

The *Challenge Content Creator* (see Figure 4) and the *Challenge Manager* (see Figure 5) pages are directed at challenge creation and editing. The challenge content creator allows the creation of new initial board states and new code snippets. The challenge manager lists all existing challenges, allowing a teacher to edit the challenge. To create a brand-new challenge, there is the *Create Challenge* button.


Challenge Manager

Create Challenge

Test File 1: Is Move Valid?

Challenge 1.1 Passed ✓
Checking for Out of Bounds
Pieces can never leave the board.
100pts
statement
Very Easy

Challenge 1.2
Thorough Checking of Out of Bounds
What if pieces could go out of bounds?
200pts
decision
Easy

Challenge 1.3
Focus on the Destination
In this challenge, only the destination is tested, as thoroughly as possible.
500pts
mcdc
Hard

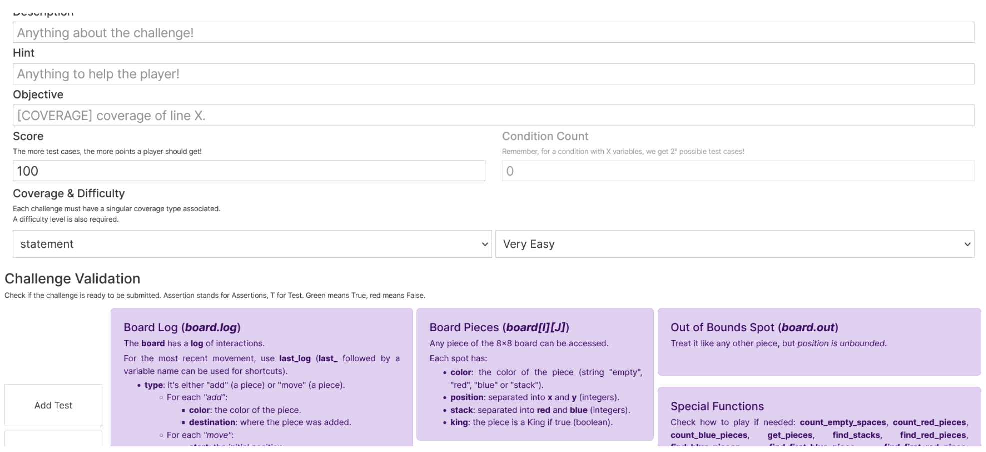
Challenge 1.4
(Not) Moving Diagonally
Checkers only allows for diagonal movement. You need to avoid that, in this challenge.
100pts
statement
Very Easy

Fig. 5: *Challenge Manager* page.

The challenge creator is divided into three sections. The first Section asks for the code file and initial board state to associate with the challenge. The second Section (see Figure 6) asks for the challenge details (title, description, objective, score, coverage type, difficulty, and, if relevant, condition count). The third Section asks for the challenge's evaluation expressions.

Evaluation expressions are abstracted for user-friendliness and follow a specific format. The third Section (see Figure 7) gives instructions on how to write these expressions with proper JavaScript syntax. As an example of the abstraction, "board" is internally translated into the proper indexation expression to access the board when the challenge is submitted to the database.

Expressions can be assertions or tests. Assertions are optional and improve the performance of the evaluation process, while the tests define the criteria for the success of the challenges. To successfully pass a challenge, the user must pass its assertions and tests. To finish creating a challenge, the user must pass their proposed challenge, thus


Anything about the challenge!

Hint

Anything to help the player!

Objective

[COVERAGE] coverage of line X.

Score

The more test cases, the more points a player should get!

100

Condition Count

Remember, for a condition with X variables, we get 2^x possible test cases!

0

Coverage & Difficulty

Each challenge must have a singular coverage type associated.
A difficulty level is also required.

statement

Very Easy

Challenge Validation

Check if the challenge is ready to be submitted. Assertion stands for Assertions, T for Test. Green means True, red means False.

Board Log (***board.log***)

The **board** has a **log** of interactions.

For the most recent movement, use **last_log** (**last_** followed by a variable name can be used for shortcuts).

- **type**: it's either "add" (a piece) or "move" (a piece).
    - For each *"add"*:
        - **color**: the color of the piece.
        - **destination**: where the piece was added.
    - For each *"move"*:

Board Pieces (***board[I][J]***)

Any piece of the 8×8 board can be accessed.

Each spot has:

- **color**: the color of the piece (string "empty", "red", "blue" or "stack").
- **position**: separated into **x** and **y** (integers).
- **stack**: separated into **red** and **blue** (integers).
- **king**: the piece is a King if true (boolean).

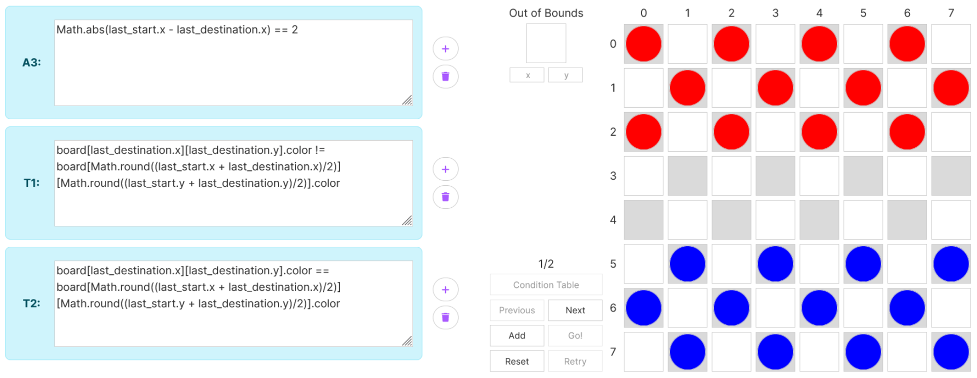
Out of Bounds Spot (***board.out***)

Treat it like any other piece, but *position is unbounded*.

Special Functions

Check how to play if needed: **count_empty_spaces**, **count_red_pieces**, **count_blue_pieces**, **get_pieces**, **find_stacks**, **find_red_pieces**,

Fig. 6: *Challenge Creator* page, 2nd Section.


A3: Math.abs(last_start.x - last_destination.x) == 2

T1: board[last_destination.x][last_destination.y].color != board[Math.round((last_start.x + last_destination.x)/2)][Math.round((last_start.y + last_destination.y)/2)].color

T2: board[last_destination.x][last_destination.y].color == board[Math.round((last_start.x + last_destination.x)/2)][Math.round((last_start.y + last_destination.y)/2)].color

Fig. 7: *Challenge Creator* page, 3rd Section (*Expression Maker*).

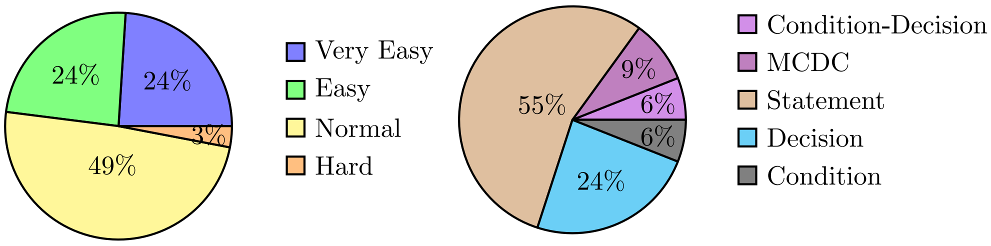
verifying that it is solvable, which unlocks the *Submit* button. Clicking this button makes the challenge visible to students.

3.3 Content

A total of 33 challenges have been developed, with the distribution across difficulty and coverage as shown in Figure 8.

As mentioned before, challenges include an associated code snippet and each code snippet can be associated with multiple challenges. These code snippets may be of varying length, which may create different objectives to create challenges around.

Next, we will provide a short overview of the two files that are provided as code snippets in the game's base challenges.



(a) Distribution of difficulty. **(b)** Distribution of coverage (`challenge_type`).

Fig. 8: Summary of tutorial challenges.

3.3.1 Code File 1: Is Move Valid

Code File 1 (in Listing 2) is the function *`is_valid_move`* checking the validity of a piece movement. The rules of *Checkers* dictate the following possible moves: making diagonal moves toward the opponent's side to an empty place and making a diagonal move to an empty place over an opponent's piece (capture). The function tests this by checking the movement's *start* and *destination* spots and the difference between their horizontal (rows) and vertical (columns) coordinates.

In total, 19 challenges (titled in a 1.X format) were made for this code file.

```
// board includes the board's state.
// start: the (x, y) position from where a piece last moved.
// destination: the (x, y) position to where a piece last moved.
function is_valid_move(board, start, destination) {
  // Math.abs() gives you the absolute value of whatever you call it with.
  // Math.abs(-1) == 1
  if (destination.x < 0 || destination.x > 7 ||
    destination.y < 0 || destination.y > 7) {
    return false;
  }

  var lineDifference = Math.abs(start.x - destination.x);
  var columnDifference = Math.abs(start.y - destination.y);

  if (lineDifference != columnDifference) {
    return false;
  } else {
    if (lineDifference == 1) {
      if (board[destination.x][destination.y].color == Color.EMPTY) {
        return true;
      } else {
        return false;
      }
    } else if (lineDifference == 2) {
      if (board[destination.x][destination.y].color == Color.EMPTY) {
        var middlePiece = board[Math.round((start.x + destination.x) / 2)][Math.round((start.y +
            destination.y) / 2)];
        if (middlePiece.color == board[start.x][start.y].color) {
          return false;
        } else if (middlePiece.color != Color.EMPTY) {
          return true;
        }
      } else {
        return false;
      }
    } else {
      return false;
    }
  }
}
```

Listing 2: Code File 1 content.

3.3.2 Code File 2: Is Board Valid

The *is_board_valid* function, presented in Code File 2 (in Listing 3), determines the validity of a *Checkers* board. This function checks the quantity and positions of pieces. In a *Checkers* game, the pieces are placed on black squares. Thus, for a board to be valid, there cannot be pieces in an even row and odd column position and vice versa. Furthermore, there must be at least one piece of each color and, at most, twelve pieces of each color.

GAMFLEW has 15 challenges based on this code file, with an identifier in the format 2.*X*.

```
function is_board_valid(board) { // board is the board's state.
  var bluePieces = this.count_blue_pieces(board);
  var redPieces = this.count_red_pieces(board);

  if (bluePieces > 12 || redPieces > 12) {
    return false;
  } else if (bluePieces == 0 || redPieces == 0) {
    return false;
  }

  var pieces = this.get_pieces(board);
  var odd = [1, 3, 5, 7], even = [0, 2, 4, 6];

  for (p in pieces) {
    if (p.position.x % 2 != 0) {
      if (!odd.includes(p.position.y)) {
        return false;
      }
    } else {
      if (!even.includes(p.position.y)) {
        return false;
      }
    }
  }

  return true;
}
```

Listing 3: Code File 2 content.

4 Preliminary Evaluation

To validate *GAMFLEW*, we designed an evaluation experiment. We recruited participants through our network (e.g., past students). Our questionnaires were implemented on Google Forms and shared online. Per our inclusion criteria, participants were required to be at least 18 years old, and students or graduates in a topic related to informatics engineering, at BSc level or above.

The subsequent sections describe the *GAMFLEW* validation study performed on March 14, 2024 in a remote manner, via Microsoft Teams. Its script is as follows, totaling 2h15m:

- **Step 1:** Pre-questionnaire (10 minutes) to assess personal characteristics.
- **Step 2:** Introduction to Code Coverage Concepts (15 minutes).
- **Step 3:** Game Installation (15 minutes).
- **Step 4:** Presenting the Game (10 minutes).
- **Step 5:** Playing the Game (60 minutes).
- **Step 6:** Short Exam (15 minutes).
- **Step 7:** Post-questionnaire (10 minutes)

4.1 Study Goals and Research Questions

The goals of the experiment were the following:

- **Objective 1:** Collect players' feedback to guide the development of new features or improvements to *GAMFLEW*.
- **Objective 2:** Assess the impact of *GAMFLEW* on player learning and understanding of white-box testing concepts.

With this in mind, the research questions we intend to address are:

- **RQ1:** How do players react to the game and which features would they most like to see added? (**Objective 1**)
- **RQ2:** What is *GAMFLEW*'s impact in the student understanding of the white-box testing concepts it features? (**Objective 2**)

4.2 Considerations

Based on the previous subsection, we considered the following aspects in the experiment design:

- As *GAMFLEW* presents JavaScript code, assessing players' knowledge of JavaScript is important.
- For a player with more programming experience, we assume that it will be easier to understand the code presented, making it important to evaluate the programming experience.
- Although the test cases are submitted through interaction with the Checkers board, knowing the rules of this game is not essential because the interaction is much more flexible, allowing movements of pieces that are not allowed in a Checkers game.
- As *GAMFLEW* does not explicitly teach white-box testing concepts, assessing whether players have this prior knowledge is essential.

4.3 Experiment Description

The experiment was structured into eight steps, which we explore next.

Anonymity

As per **Objective 2** of the experiment, it is important to evaluate how the experiment's participants were impacted by using *GAMFLEW*. However, many steps were taken towards the anonymization and protection of the participants' identities.

The data collected can be considered purely demographic or insufficient to allow users to be identified. In the interest of following the best practices in our investigation, participants were free to abandon the experiment at any point and were made aware of our commitment to their anonymity via disclaimers that preceded all questionnaires.

Users were asked to provide their own three-letter and two-number code — for example, *LJD24* — to use in all moments of data collection, thus allowing our analysis to trace each user's performance and analyze it against their stated (relevant) characteristics, such as programming experience, but without revealing the identity of the participants.

4.3.1 Step 1: Pre-questionnaire

Before introducing the game, we collected all relevant information about the participants. The following questions were part of the pre-questionnaire:

- Their unique code for the experiment (as mentioned in Anonymity).
- Personal characteristics:
  - Age range, restricted to four mutually exclusive ranges.
  - Self-description, with the possibility to answer however desired (for example, *Male*).
  - Course acronym (for example, *MEIC* — Masters Degree in Informatics and Computing Engineering).
  - Course year, restricted to *"1st"*, *"2nd"* or *"3rd"*.
- Technological skills:
  - Previous knowledge in software testing, rated on a 5-level range going from *"No knowledge"* to *"Extensive knowledge"*.
  - Previous knowledge of software testing concepts, comprising a list of mainly white-box testing concepts.
  - Years of programming experience, restricted to one of six possible answers.
  - Knowledge of the JavaScript programming language, rated on a 5-level range.
  - Confidence regarding technological skills, rated from *"Strongly disagree"* to *"Totally confident"* in a 5-level range. Includes four statements regarding confidence in using a brand-new software application.
    - **Q0.1 -** *"I could use a new software application even if I never used an application like it before."*
    - **Q0.2 -** *"I could use a new software application well if I had just the built-in-help facility or manual for assistance."*
    - **Q0.3 -** *"I could use a new software application well if I had first seen someone else using it before trying it myself."*
    - **Q0.4 -** *"I could use a new software application well using only the Internet for assistance."*

4.3.2 Step 2: Introduction to Code Coverage Concepts

At the time of the experiment, *GAMFLEW* did not explain the code coverage concepts. Instead, it allowed for their consolidation after learning about them. With this in mind, we had a short lesson with a brief introduction of the five code coverage concepts used by the game to give the participants all the theoretical knowledge needed to tackle challenges properly.

4.3.3 Step 3: Game Installation

The game was shared with participants using Docker. This allowed each participant to have their local iteration of the game. In practice, this meant that information from participants was not stored in a centralized manner during the experiment, hence the inclusion of Step 6.

With the *Docker-ized* version of the game, the users got a user manual, a README file, and a short guide for the experiment. They were also urged to comment on most, if not all, of their attempts while trying challenges.

4.3.4 Step 4: Presenting the Game

The game was introduced to the users, explaining the core mechanics of how to play it and how they relate to the previously described code coverage techniques. All single-player mechanics were presented to the participants, with the possibility to clear any doubts that arose.

4.3.5 Step 5: Playing the Game

The users were then asked to open the game. For the experiment, five challenges were chosen as recommended for the users to prioritize. The main objective for participants was to pass these five challenges, one of each of the code coverage types included,[7] and afterwards try to solve as many other challenges as possible within the time limit.

Users were given 60 minutes to play the game. Beyond potential assistance with unpredictable errors, they were left to play autonomously.

4.3.6 Step 6: Data Collection

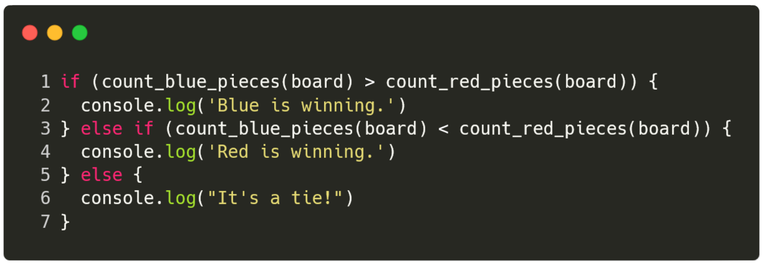
To circumvent the non-centralized manner in which the users played the game, users were instructed to download a dump of their attempts from the prototype's database using *pgAdmin*'s web interface, which was included in the provided Docker version of the game. These dumps were anonymously submitted using a file drop — participants were asked to rename them using the unique code they created in the pre-questionnaire.

4.3.7 Step 7: Short Exam

After playing the game, participants answered a short exam. It aimed to assess knowledge about the theoretical concepts of the game. The exam questions were:

[7]The challenges chosen were Challenge 1.1, Challenge 1.2, Challenge 2.4, Challenge 2.7, and Challenge 2.8.

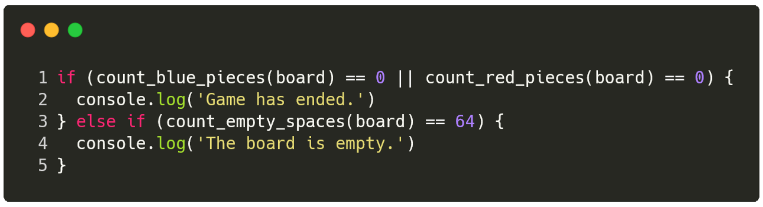
- **Questions 1.1:** In a statement coverage challenge to cover a specific line of code, do we need more than one test case to overcome the challenge?
- **Question 1.2:** Justify your previous answer.
- **Question 2:** Determine the minimum number of test cases needed to achieve 100% decision coverage of line 1 in Figure 9. Answer options: *"one"*, *"two"*, or *"three"*.



```
1 if (count_blue_pieces(board) > count_red_pieces(board)) {
2   console.log('Blue is winning.')
3 } else if (count_blue_pieces(board) < count_red_pieces(board)) {
4   console.log('Red is winning.')
5 } else {
6   console.log("It's a tie!")
7 }
```

Fig. 9: *Short Exam, Question 2* Code Snippet

- **Question 3:** Determine the minimum number of test cases necessary to achieve 100% condition coverage for line 3 in Figure 10. Options were: *"1 - because the Boolean expression has only one condition"*; *"2 - because the condition must be covered for cases True and False"*; and *"3 - because there are three determinant test cases"*.



```
1 if (count_blue_pieces(board) == 0 || count_red_pieces(board) == 0) {
2   console.log('Game has ended.')
3 } else if (count_empty_spaces(board) == 64) {
4   console.log('The board is empty.')
5 }
```

Fig. 10: *Short Exam, Question 3* Code Snippet

- **Question 4:** Identify the correct set of test cases needed to achieve 100% modified condition/decision coverage for line 1 in Figure 10. Answer options were: *"*$\{1, 2, 3, 4\}$*"*, *"*$\{1, 2, 3\}$*"*, *"*$\{1, 2\}$*"*, and *"*$\{2, 3, 4\}$*"*.

4.3.8 Step 8: Post-questionnaire

Finally, participants were asked to fill out a post-questionnaire (Appendix A).

The post-questionnaire has nine different sections, with statements to be rated by the participants on a standard Likert scale (going from *Strongly disagree* to *Strongly agree*), each regarding different aspects of *GAMFLEW*.

The sections are as follows:

- **Section 1 -** ***Ease of use and usability***;

  - **Q1.1 -** "*GAMFLEW* is easy to use/follow."
  - **Q1.2 -** "*GAMFLEW* is well organized so it is easy to find the necessary information."
- **Section 2 -** ***Learnability***: the ease with which users get familiar with using the capsule;
  - **Q2.1 -** "I would imagine most of the students would start learning with *GAMFLEW* quickly."
  - **Q2.2 -** "*GAMFLEW* is clear and understandable."
  - **Q2.3 -** "I need a lot of background knowledge to be able to learn with *GAMFLEW*."
  - **Q2.4 -** "I would need teacher support to start learning with *GAMFLEW*."
- **Section 3 -** ***Perceived satisfaction, confidence, and comfort***;
  - **Q3.1 -** "Overall, I am satisfied with learning based on *GAMFLEW*."
  - **Q3.2 -** "I felt confident learning with *GAMFLEW*."
  - **Q3.3 -** "I felt comfortable learning with *GAMFLEW*."
  - **Q3.4 -** "*GAMFLEW* motivated me to learn."
  - **Q3.5 -** "I would recommend *GAMFLEW* to other students."
- **Section 4 -** ***Personal difficulties or complexity***;
  - **Q4.1 -** "I found *GAMFLEW* unnecessarily complex."
  - **Q4.2 -** "I thought there was too much inconsistency in *GAMFLEW*."
  - **Q4.3 -** "I found *GAMFLEW* very cumbersome."
  - **Q4.4 -** "Learning with *GAMFLEW* is a frustrating experience."
  - **Q4.5 -** "I have spent too much time correcting things in *GAMFLEW*."
- **Section 5 -** ***Perceived effectiveness, productiveness, and efficiency***;
  - **Q5.1 -** "I can effectively learn with *GAMFLEW*."
  - **Q5.2 -** "I can improve my learning performance using *GAMFLEW*."
  - **Q5.3 -** "I can improve my learning productivity using *GAMFLEW*."
  - **Q5.4 -** "It is easier to learn with *GAMFLEW*."
  - **Q5.5 -** "The answer you should provide here is Disagree." [8]
- **Section 6 -** ***Overall usefulness perception (fit for purpose)***;
  - **Q6.1 -** "*GAMFLEW* meets my learning requirements."
  - **Q6.2 -** "*GAMFLEW* has all the content I expect it to have."
  - **Q6.3 -** "I would like to use *GAMFLEW* frequently."
  - **Q6.4 -** "I would find *GAMFLEW* useful in my learning process."
  - **Q6.5 -** "It would be easy for me to become more qualified using *GAMFLEW*."
- **Section 7 -** ***System supporting information and feedback***;
  - **Q7.1 -** "The information (such as online help, on-screen messages, etc) is helpful."
  - **Q7.2 -** "The information is effective in helping me complete the learning tasks."
  - **Q7.3 -** "The organization of information on the *GAMFLEW* screens is clear."
  - **Q7.4 -** "*GAMFLEW* gives error messages that clearly tell me how to correct the problems."
  - **Q7.5 -** "Whenever I make a mistake using *GAMFLEW*, I recover easily."

[8] First wellness check, to avoid influencing participants and as a validation of their feedback.

- **Section 8 - *System Interface and interaction***;
  - **Q8.1** - "The answer you should provide here is Neutral." [9]
  - **Q8.2** - "The interface of the *GAMFLEW* is pleasant."
  - **Q8.3** - "I like using the interface of *GAMFLEW*."
  - **Q8.4** - "The interaction with *GAMFLEW* is clear and understandable."
- **Section 9 - *Serious Game (game experience, usability, and learning experience***;
  - **Q9.1** - "The experience was challenging and stimulating."
  - **Q9.2** - "I was able to achieve the goals set in *GAMFLEW*."
  - **Q9.3** - "I remained focused on *GAMFLEW* throughout."
  - **Q9.4** - "It was easy to respond to in-game *GAMFLEW* survey questions."
  - **Q9.5** - "The learning goals of the *GAMFLEW* game were clear."
  - **Q9.6** - "*GAMFLEW* provided opportunities to receive feedback."
  - **Q9.7** - "I recognize the value of *GAMFLEW* as a tool for learning."

At the end of the post-questionnaire, users could provide comments or feedback they wanted to share in a final question with a free text box.

4.4 Results & Discussion

The experiment results are described in the following sections.

4.4.1 Pre-questionnaire

Participants

There were nine participants in total, which were labelled using uppercase letters. On their demographic distribution:

- Seven participants identified as Male, and the remaining two identified as Female.
- Seven participants were in the 18-25 age range, while the remaining two were in the 25-35 age range.
- Seven participants are in a Masters program, one participant is in a Doctorate program and the final participant provided a null answer.
- Seven participants are in the 2nd year of their course, one in their 1st year, and the final participant provided a null answer.

The vast majority of participants were Male. Regarding age, most were between 18 to 25 years of age, which would be the age at which students study for a Master's degree. This is also reflected in the academic course distribution. Participant H answered neither of the prior courses, signaling they'd finished their studies. Most participants, all Master students, were in their second year, while the PhD student (Participant B) was in their 1st year. Once again, there's a null answer.

Technological Skills

Regarding technological skills, the participants stated that the average software testing knowledge (3rd question) was 3.11, where level 3 is taken as average knowledge. Participant H provided the only "Extensive knowledge" answer regarding software testing. Most students answered with level 2 (below average) or level 3. Regarding

[9] Second wellness check, to avoid influencing participants and as a validation of their feedback.

Question 2, the most well-known concepts were unit testing, mutation coverage, white-box testing, black-box testing, and statement coverage. The latter is assumed to be known because of code coverage when applied to unit testing environments. Question 2's results are summarized in Figure 11.

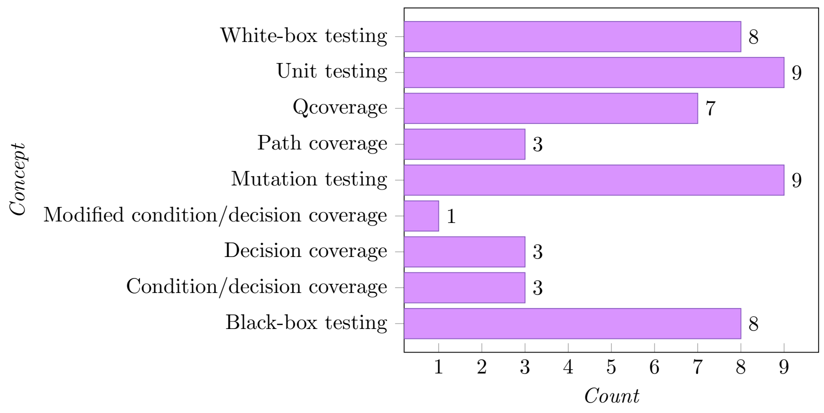


Fig. 11: Second question (known software testing concepts) summary.

Programming experience was mostly in the 4-6 years range (6 participants), followed by 1-3 years (2 participants, one of them Participant H) and 7-9 years (Participant F). The average knowledge of the JavaScript programming language was 3.22, where level 3 is considered the average knowledge. Most participants answered on levels 3 and 4 (above average), with Participant H being the only one claiming not to know JavaScript.

Ultimately, on the four statements regarding technological skills (Figure 12), the results show that all participants are confident in their capability to use a new software tool — only **Q0.1** had 3 neutral answers, with all other answers being above average confidence (*Likely yes* or *Totally confident*).

4.4.2 Attempts

From the files submitted by each participant, a summary of the total attempts was made (shown in Table 1). Only Participant H did not complete the 5 recommended challenges in the given time. In general, most participants felt compelled to go beyond the recommended challenges, though to varying degrees. Only Participant H did not finish the recommended challenges, while all other participants (except Participants F and I) completed at least 2 more attempts. In fact, Participant B almost completed all existing 33 challenges (33 passed attempts, 32 different challenges), followed by Participant G (28 passed attempts, all for different challenges), and Participants C and D (24 passed attempts, with 24 and 23 different challenges, respectively).

This leads us to believe that *GAMFLEW* provided a motivating environment for learning the presented concepts.

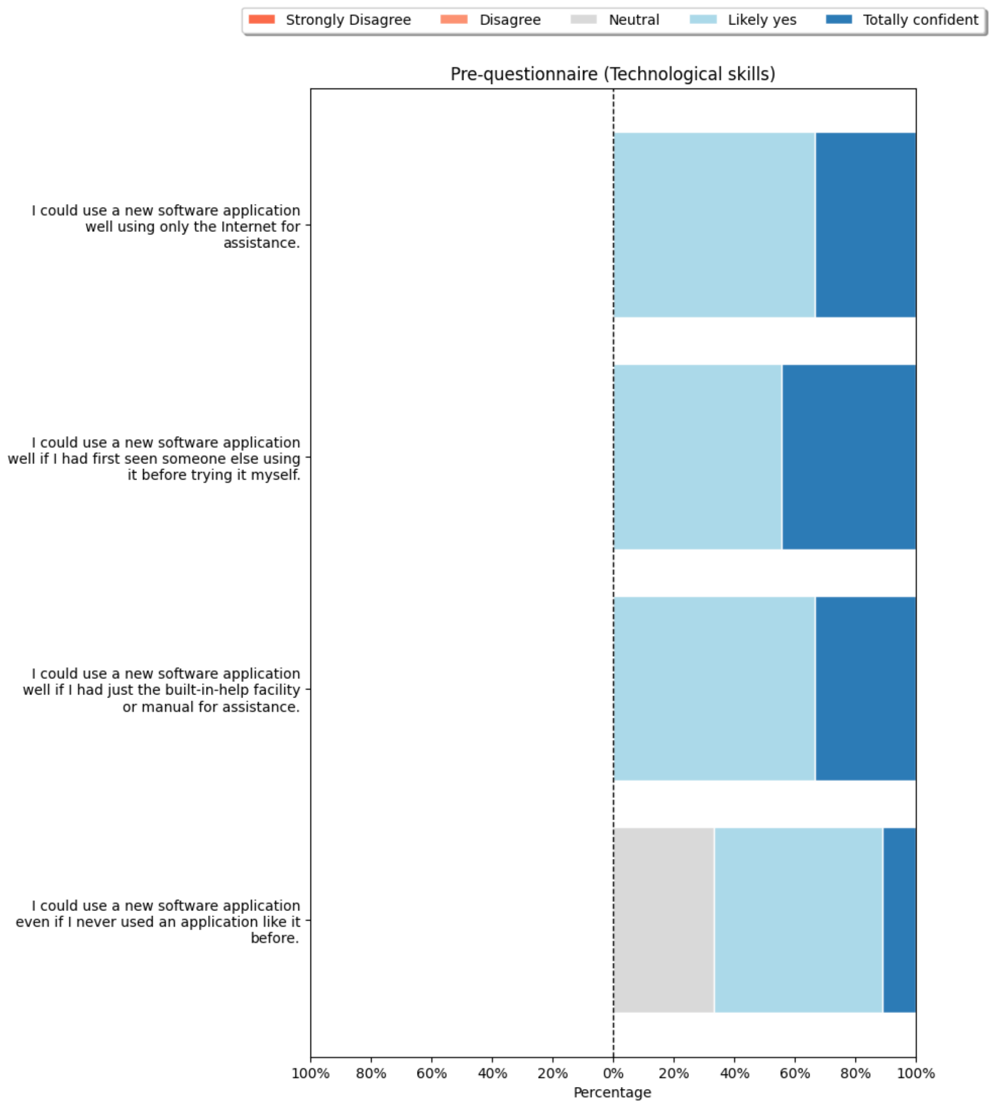


Fig. 12: Pre-questionnaire results (Technological Skills).

4.4.3 Short Exam

In all multiple-choice questions, with the exception of the 2nd question, there was only 1 wrong answer. In the 2nd question, there were two wrong answers.

Regarding question 1.2, all justifications were correct, even the one from the supposedly wrong answer, stating that *it may depend on the number of code lines in a challenge*. This is considered correct, though no existing challenge asks to achieve 100% statement coverage of more than one line. Table 2 summarizes the results.

Table 1: Attempts summary by participant.

Participant	Pass	Fail
A	7	0
B	33	4
C	24	1
D	24	0
E	9	2
F	5	1
G	28	0
H	4	0
I	5	0

Table 2: Short exam results.

	Question 1.1	Question 2	Question 3	Question 4
Correct	8	7	8	8
Wrong	1	2	1	1

When studying the answers, it was postulated that all the wrong answers were from the same participant, which was not true. For most questions, each incorrect answer came from a different participant — Participant G got question 1 wrong, and Participant D got question 2 wrong. The mistake may have spanned from a simple misunderstanding of the code coverage measures as introduced at the beginning of the experiment.

More interestingly, Participant H got questions 2, 3, and 4 wrong — which may be tentatively explained with their 1-3 years of programming experience and no knowledge of JavaScript. Especially the latter is considered key to do well in the game's challenges, as understanding a challenge's code snippet (written in JavaScript) is required to know how to beat the challenge.

These promising results lead us to believe participants may have benefited from using *GAMFLEW* to learn the presented concepts.

4.4.4 Post-Questionnaire

The following sections summarize each of the results of the post-questionnaire sections. For the sake of simplicity, averages are used to convey the results in an aggregated manner.

Ease of use and usability

The most negative answers regarding *GAMFLEW*'s ease of use and organization were both *Neutral*. All remaining answers agreed, with a tendency towards *Agree* (4th level). **Q1.1**'s average was 4.22, while **Q1.2** gave an average of 4.11.

Figure 13 summarizes the results which indicate that *GAMFLEW*'s interface is responsive and easy to use, but with some room for improvement.

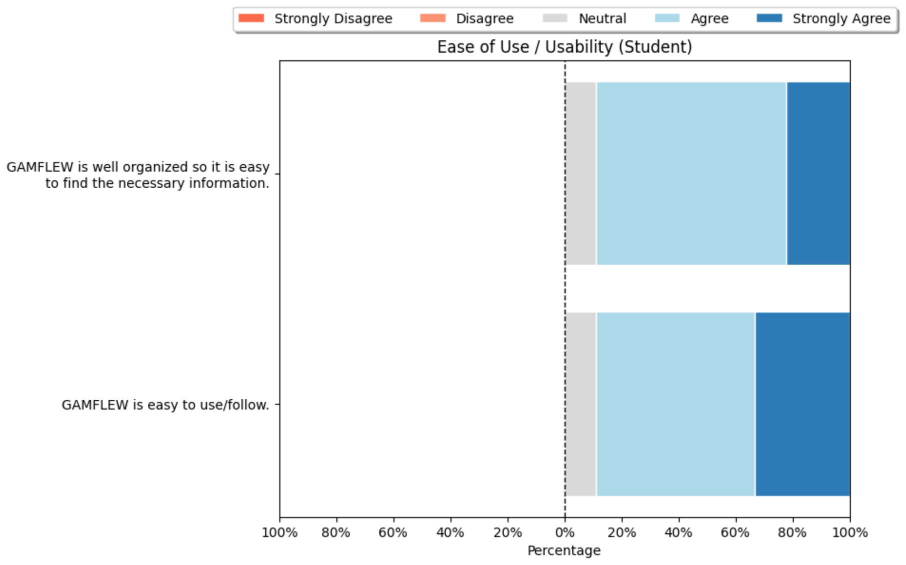


Fig. 13: Post-questionnaire results (Section 1).

Learnability

Regarding learnability, participants agree that players can quickly learn with *GAMFLEW* (4.00 average in **Q2.1**) and that it is clear and understandable (3.98 average in answers to **Q2.2**). Participants do not need much background knowledge to play *GAMFLEW* (2.33 average in **Q2.3**) nor need much teacher support to start learning with *GAMFLEW* (2.56 average in **Q2.4**).

Figure 14 summarizes the results, that appear to show that *GAMFLEW*'s supports the learning process.

Perceived satisfaction, confidence, and comfort

There is overall satisfaction with *GAMFLEW* (4.22 average in **Q3.2**), meaning that participants felt confident when learning through gameplay. Also, participants felt comfortable and motivated (4.78 in both **Q3.3** and **Q3.4**) and would mostly recommend *GAMFLEW* to others (4.56 average in **Q3.5**).

Results shown in Figure 15 lead us to believe that *GAMFLEW* motivates and captivates students attention.

Personal difficulties and complexity

Regarding difficulties and complexity, participants think that *GAMFLEW* is not complex (2.00 average in **Q4.1**), it is consistent (1.44 average in **Q4.3**), and it is not cumbersome (2.22 average in **Q4.3**). Participants agreed that *GAMFLEW* provides a non-frustrating experience (1.33 average in **Q4.4**) and did not spend a lot of time correcting things (1.67 average in **Q4.5**).

Figure 16 summarizes the results, which suggest that *GAMFLEW* is not complex.

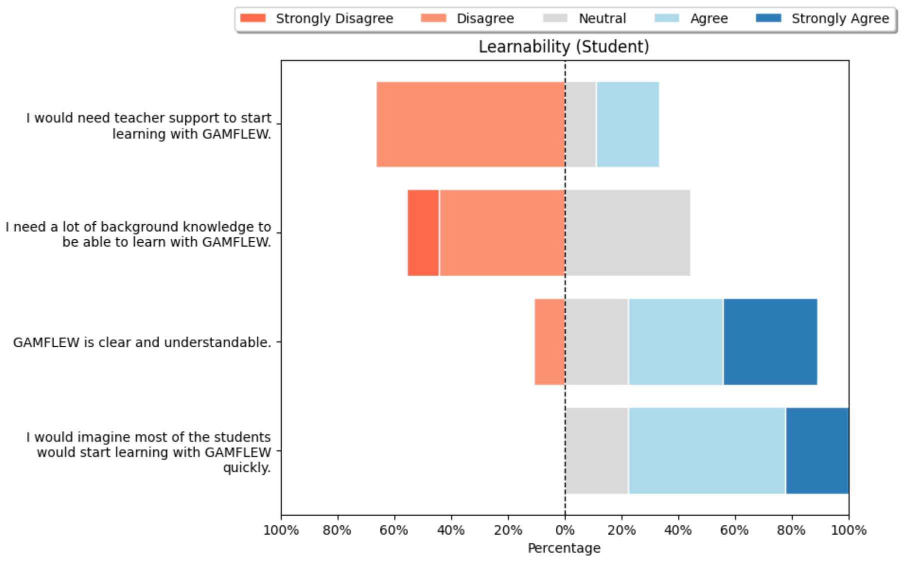


Fig. 14: Post-questionnaire results (Section 2).

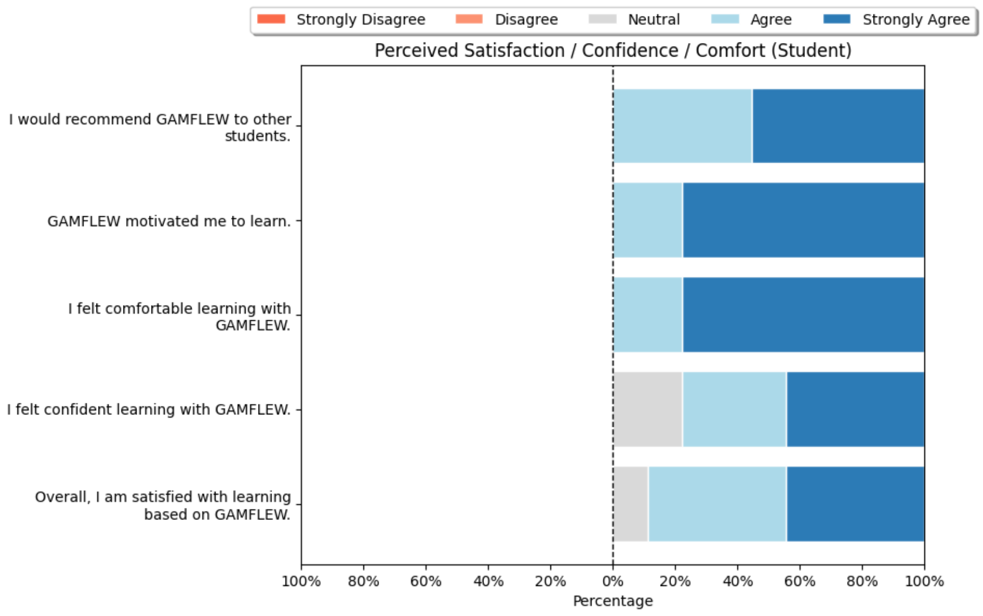


Fig. 15: Post-questionnaire results (Section 3).

Perceived effectiveness, productiveness, and efficiency

Participants felt they could effectively learn with *GAMFLEW* (4.44 average in **Q5.1**) and could improve their learning performance (4.44 average in **Q5.2**) and productivity

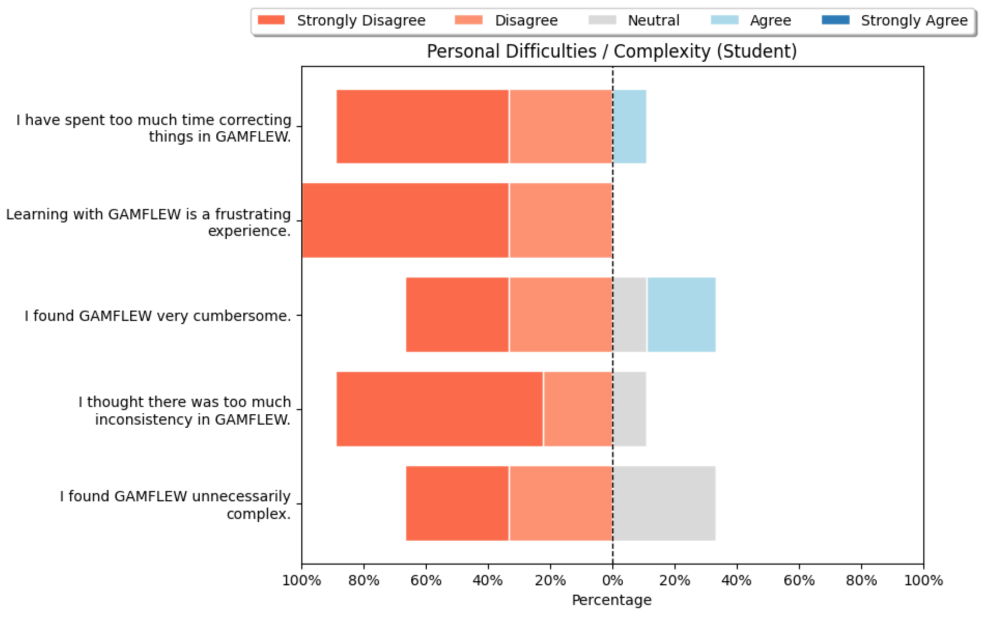


Fig. 16: Post-questionnaire results (Section 4).

(4.22 average in **Q5.3**). In addition, results show that participants believe it is easier to learn with *GAMFLEW* (4.22 average in **Q5.4**). Finally, all participants correctly answered the attention check question.

Figure 17 summarizes the results, which lead us to believe that *GAMFLEW* promotes learning among players.

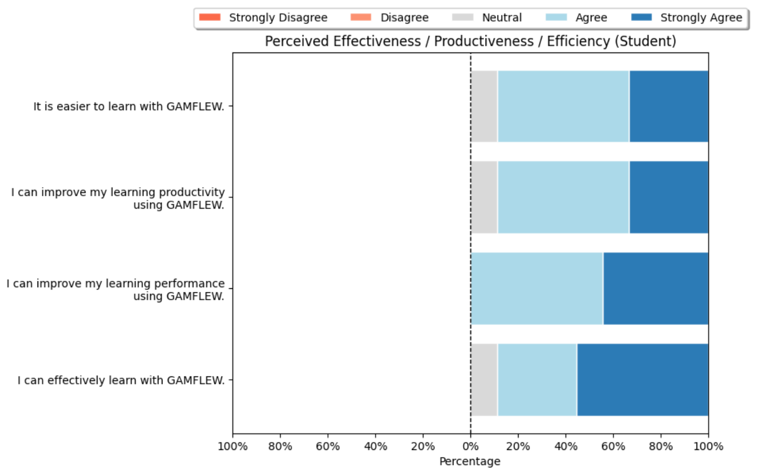


Fig. 17: Post-questionnaire results (Section 5).

Overall usefulness perception (fit for purpose)

Participants believed *GAMFLEW* met their learning requirements (4.11 average in **Q6.1**) and slightly agree that *GAMFLEW* had all the content they expected (3.89 average in **Q6.2**). Participants wish to use *GAMFLEW* frequently (3.67 average in **Q6.3**), believe it is useful to their learning process (4.67 average in **Q6.4**), and believe it is helpful to become more qualified (4.11 average in **Q6.5**).

The results in Figure 18 suggest that *GAMFLEW* promotes and is useful in the learning process.

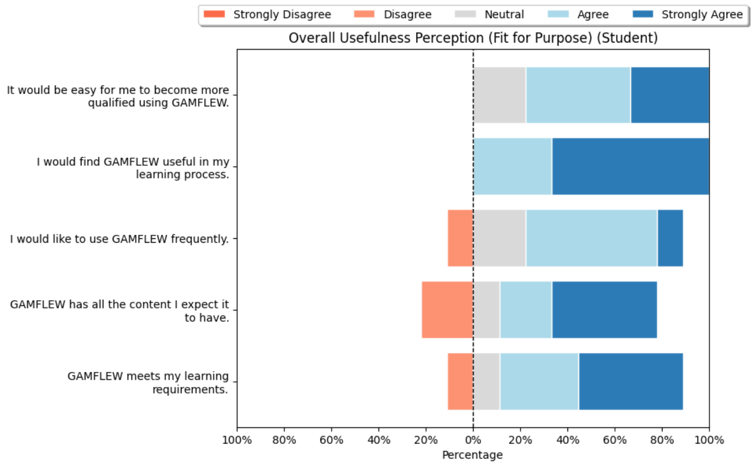


Fig. 18: Post-questionnaire results (Section 6).

System supporting information and feedback

The information provided by *GAMFLEW* to the players is helpful (4.11 average in **Q7.1**), effective (3.89 average in **Q7.2**), and clearly organized (4.00 average in **Q7.3**). Error messages communicate corrective actions effectively (3.44 average in **Q7.4**), and there is a general agreement that it is easy to recover from mistakes (4.33 average in **Q7.5**).

The results in Figure 19 suggest that the game's interface is clear and mostly intuitive.

System Interface and interaction

All participants passed the second and final wellness check. In general, participants found *GAMFLEW*'s interface pleasant (average 4.44 - **Q8.2**), enjoyed using it (average 4.56 - **Q8.3**), and found interactions to be clear and understandable (average 4.11 - **Q8.4**).

Figure 20 shows the results, which seem to indicate an user interface positively received by players.

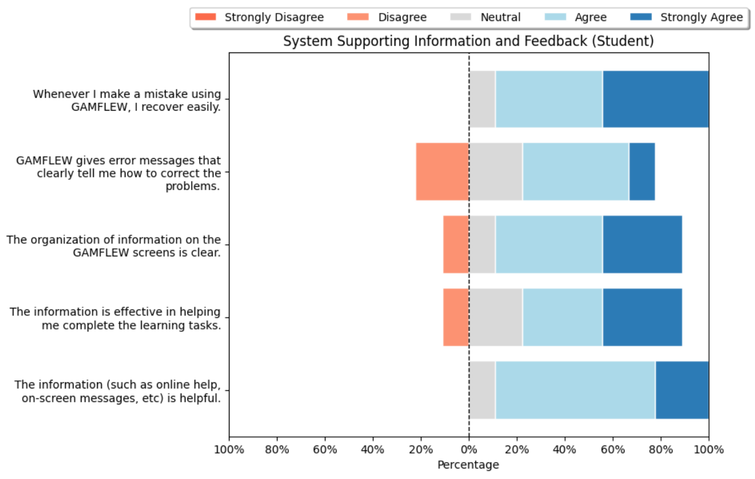


Fig. 19: Post-questionnaire results (Section 7)

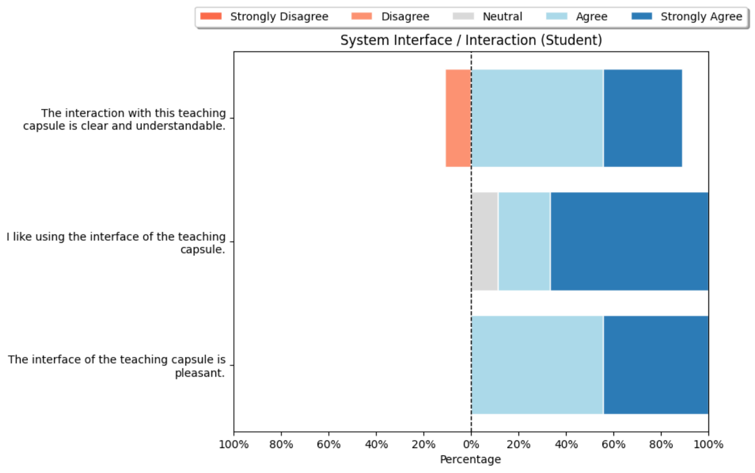


Fig. 20: Post-questionnaire results (Section 8)

Serious game (game experience, usability, and learning experience)

Participants found the experiment challenging and stimulating (4.11 average on **Q9.1**). They generally believed they achieved the goals set in *GAMFLEW* (4.56 average in **Q9.2**). Participants remained focused throughout the experiment (4.11 average in **Q9.3**) and questions were easy to answer (4.11 average in **Q9.4**). Participants also found *GAMFLEW*'s learning goals clear (average 4.33 on **Q9.5**) and agreed to have chances to provide feedback (average 4.00 on **Q9.6**). Finally, participants strongly recognized *GAMFLEW*'s value as a tool for learning (4.78 average in **Q9.7**).

We believe that the results shown in Figure 21 indicate that *GAMFLEW* provides a pleasant and helpful environment for learning.

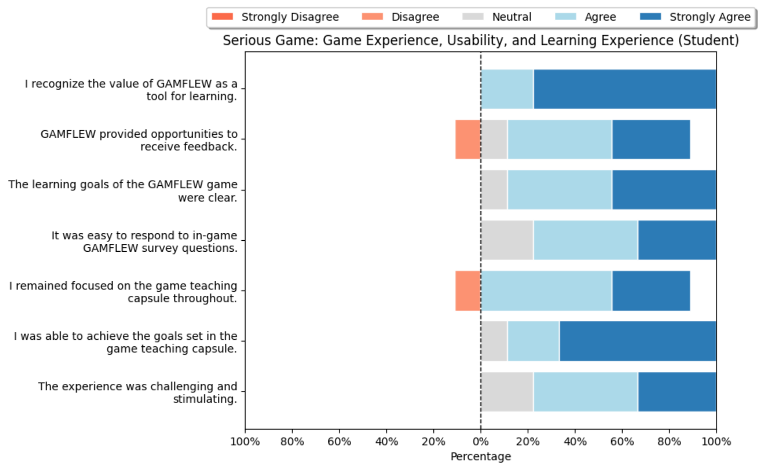


Fig. 21: Post-questionnaire results (Section 9).

In general, the post-questionnaire provided positive results. However, considering the comments submitted in this questionnaire, *GAMFLEW* still has room to improve. One of the more frequent suggestions asked for in-game explanations of the code coverage measures it features. Other comments pointed out occurrences or feedback for further development of the game's usability, such as a back arrow present on all pages and the saving of a user's scroll position in the challenge selection menu.

Beyond this, comments complimented the game's appearance and structure and provided constructive feedback on certain occurrences that could be classified as bugs. No player reported a game-breaking bug, which points towards the robustness of the game and its design.

4.4.5 Answers to the Research Questions

RQ1: How do players react to the game and which features would they most like to see added?

The results appear to indicate that players' reactions to the game are positive, as participants enjoyed its interface, gameplay, and associated features. In general, they asked for usability-related features, such as a dedicated back button and in-game explanations of the code coverage measures for their easy access.

RQ2: What is GAMFLEW's impact in the student understanding of the white-box testing concepts it features?

Although the results of our preliminary evaluation are not generalizable, due to the number of participants, the results seem to indicate that *GAMFLEW* has positively impacted the participants' understanding of white-box testing concepts. Though many participants claimed to have heard of the featured code coverage measures before, the game may have helped consolidate their knowledge or impacted their understanding of such concepts. This analysis is derived from both the confidence the participants admitted in the post-questionnaire and the overall positive results of the short exam.

4.5 Limitations and Threats to Validity

At the start of the experiment, we asked participants to self-assess their knowledge and technological skills. At the end, we asked students to perform a short exam to assess their knowledge. We acknowledge this asymmetrical evaluation, i.e., self-evaluation may be subjective while the final evaluation is not. However, we made this choice to avoid giving an exam at the start, which could reveal the specific knowledge we expected them to gain during gameplay, which could influence how they approached the game and alter their overall experience.

In addition, we are aware that, due to the number of participants in the experiment, the results cannot be considered statistically relevant and, therefore, cannot be generalized. However, the goal was to conduct a preliminary evaluation that could inform future improvements to the game.

Further threats include the Hawthorn effect, which we attempted to mitigate by not observing the users as they were carrying out the experiment and by collecting data anonymously to protect user privacy.

Beyond this, some game bugs experienced by the participants may have affected their performance. However, all feedback collected has been taken into account and will be considered in *GAMFLEW*'s future developments.

5 Conclusion

This paper presents a novel serious game to teach white-box test case design techniques, *GAMFLEW*. Following a current trend in introducing gamification techniques in education, *GAMFLEW* is another contribution in this direction tailored to educators and students.

GAMFLEW provides a new learning experience for students who try to overcome the challenges (i.e., design test cases to achieve defined coverage criteria) designed by their teachers (i.e., the code to analyze and the coverage criteria to fulfill). The results obtained in the experiment with students give us confidence in the potential of *GAMFLEW* as a tool for teaching software testing. Although there is room for improvement, the experiment has shown that *GAMFLEW* creates a positive learning experience in which participants are motivated and interested.

All evaluated aspects of the game, as specified in the experiment's post-questionnaire, gave positive results: users found *GAMFLEW* easy to use, the learnability to be adequate, showed satisfaction while using it, experienced few difficulties, recognized its effectiveness and usefulness, and found its interface informative, helpful, and pleasant. Beyond this, participants enjoyed the game and the experiment they partook in.

The results lead us to believe we are on the right track to fulfilling *GAMFLEW*'s learning goals, as conceptualized from its inception, out of which we can highlight the participation of educators as more than mediators of gameplay but as contributors to it. We are committed to bettering and continuing work on *GAMFLEW*, thus contributing to the evolution of software testing education.

5.1 Future Work

In the future, we intend to develop a multiplayer mode and repeat the validation experiment with more students to produce statistically relevant results that assess *GAMFLEW*'s impact in the learning of software testing concepts. Finally, we intend to expand the teacher-exclusive features and validate them in an experiment with teachers.

5.2 Accessing the Game

The Docker version utilized in the experiment can be accessed through the following link. The online version is available on this link.

Declarations

Funding This work is being funded by the ENACTEST Erasmus+ project number 101055874.

Conflict of interest/Competing interests The authors declare no conflict/competing interests.

Ethics approval and consent to participate All data collected in the user study was anonymous, provided voluntarily, and users could stop participating at any time, thus satisfying all the required ethical standards. At the start of the study, all participants were shown a consent form to which they had to agree prior to starting the study.

Consent for Publication Not applicable.

Data Availability Not applicable.

Materials Availability Not applicable.

Code Availability Code available here.

Author Contribution The game and study designs were discussed by all authors. Mateus Silva was mainly responsible for implementing the game under the supervision of Ana C. R. Paiva and Alexandra Mendes. All authors contributed to the final manuscript.

References

[1] Djaouti, D., Alvarez, J., Jessel, J.-P.: Classifying serious games: the g/p/s model. Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches (2011) https://doi.org/10.4018/978-1-60960-495-0.ch006

[2] Clegg, B.S., Rojas, J.M., Fraser, G.: Teaching software testing concepts using a mutation testing game. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), pp. 33–36 (2017). https://doi.org/10.1109/ICSE-SEET.2017.1

[3] Fraser, G., Gambi, A., Rojas, J.M.: 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. ECSEE '18, pp. 50–54. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3209087.3209103

[4] Oliveira, B., Afonso, P., Costa, H.: Testeg — a computational game for teaching of software testing. In: 2016 35th International Conference of the Chilean Computer Science Society (SCCC), pp. 1–10 (2016). https://doi.org/10.1109/SCCC.2016.7836022

[5] Soska, A., Mottok, J., Wolff, C.: An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education. In: 2016 IEEE Global Engineering Education Conference (EDUCON), pp. 576–584 (2016). https://doi.org/10.1109/EDUCON.2016.7474609

[6] Fraser, G.: Gamification of software testing. In: 2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST), pp. 2–7 (2017). https://doi.org/10.1109/AST.2017.20

[7] Henrique Dias Valle, P., Toda, A.M., Barbosa, E.F., Maldonado, J.C.: Educational games: A contribution to software testing education. In: 2017 IEEE Frontiers in Education Conference (FIE), pp. 1–8 (2017). https://doi.org/10.1109/FIE.2017.8190470

[8] Ribeiro, T.P.B., Paiva, A.C.R.: iLearnTest: Educational game for learning software testing. In: 2015 10th Iberian Conference on Information Systems and Technologies (CISTI), pp. 1–6 (2015). https://doi.org/10.1109/CISTI.2015.7170608

[9] Materazzo, A., Fulcini, T., Coppola, R., Torchiano, M.: Survival of the tested: Gamified unit testing inspired by battle royale. In: 2023 IEEE/ACM 7th International Workshop on Games and Software Engineering (GAS), pp. 1–7 (2023). https://doi.org/10.1109/GAS59301.2023.00008

[10] Marín, B., Vos, T.E.J., Paiva, A.C.R., Fasolino, A.R., Snoeck, M.: ENACTEST - European Innovation Alliance for Testing Education. In: CEUR Workshop Proceedings, vol. 3144 (2022). https://www.scopus.com/inward/record.uri?eid=2-s2.0-85131255272&partnerID=40&md5=23524a140850922df11d79fe0b9fbe51

[11] Marín, B., Vos, T., Snoeck, M., Paiva, A., Fasolino, A.: ENACTEST project - European innovation alliance for testing education. In: Font, J., Arcega, L., Reyes-Román, J.-F., Giachetti, G. (eds.) Proceedings of the Research Projects Exhibition Papers Presented at the 35th International Conference on Advanced Information Systems Engineering (CAiSE 2023). CEUR-WS, vol. 3413, pp. 91–96. CEUR Workshop Proceedings, Zaragoza, Spain (2023). https://ceur-ws.org/Vol-3413/paper13.pdf

[12] Ramasamy, V., Alomari, H., Kiper, J., Potvin, G.: A minimally disruptive approach of integrating testing into computer programming courses. In: 2018 IEEE/ACM International Workshop on Software Engineering Education for Millennials (SEEM), pp. 1–7 (2018)

[13] Costa, I., Oliveira, S.: A systematic strategy to teaching of exploratory testing using gamification. In: Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering. ENASE 2019, pp. 307–314. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT (2019). https://doi.org/10.5220/0007711603070314

[14] Ferreira Costa, I.E., Oliveira, S.R.B.: The use of gamification to support the teaching-learning of software exploratory testing: an experience report based on the application of a framework. In: 2020 IEEE Frontiers in Education Conference (FIE), pp. 1–9 (2020). https://doi.org/10.1109/FIE44824.2020.9273943

[15] Elgrably, I.S., Oliveira, S.R.B.: Gamification and evaluation of the use the agile tests in software quality subjects: The application of case studies. In: Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering. ENASE 2018, pp. 416–423. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT (2018). https://doi.org/10.5220/0006800304160423

[16] Gomes, R.F., Lelli, V.: Gamut: Game-based learning approach for teaching unit testing. In: Proceedings of the XX Brazilian Symposium on Software Quality. SBQS '21. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3493244.3493263

[17] Jesus, G.M., Paschoal, L.N., Ferrari, F.C., Souza, S.R.S.: Is it worth using gamification on software testing education? an experience report. In: Proceedings of the XVIII Brazilian Symposium on Software Quality. SBQS '19, pp. 178–187. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3364641.3364661

[18] Mascolo, M.: Change processes in development: The concept of coactive scaffolding. New Ideas in Psychology **23**, 185–196 (2005) https://doi.org/10.1016/j.newideapsych.2006.05.002

[19] ISTQB: ISTQB, International Software Testing Qualifications Board. https://www.istqb.org/. Accessed 28-03-2024.

[20] Bishop, J., Horspool, R., Xie, T., Tillmann, N., Halleux, J.: Code hunt: Experience with coding contests at scale, pp. 398–407 (2015). https://doi.org/10.1109/ICSE.2015.172

[21] CoderPad: Coding Games and Programming Challenges to Code Better — codingame.com. https://www.codingame.com. Accessed 04-11-2023.

[22] Elbaum, S., Person, S., Dokulil, J., Jorde, M.: Bug hunt: Making early software testing lessons engaging and affordable. In: 29th International Conference on Software Engineering (ICSE'07), pp. 688–697 (2007). https://doi.org/10.1109/ICSE.2007.23

[23] Bell, J., Sheth, S., Kaiser, G.: Secret ninja testing with halo software engineering. In: Proceedings of the 4th International Workshop on Social Software Engineering. SSE '11, pp. 43–47. Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2024645.2024657

[24] CodeSignal: (2023). https://codesignal.com/

[25] Logas, H., Whitehead, J., Mateas, M., Vallejos, R., Scott, L., Shapiro, D.G., Murray, J.T., Compton, K., Osborn, J.C., Salvatore, O., Lin, Z., Sánchez, H.A., Shavlovsky, M., Cetina, D., Clementi, S., Lewis, C.: Software verification games: Designing xylem, the code of plants. In: International Conference on Foundations of Digital Games (2014). https://api.semanticscholar.org/CorpusID:18755001

[26] Thiry, M., Zoucas, A., Silva, A.: Empirical study upon software testing learning with support from educational game, pp. 481–484 (2011)

A Appendix A: Exam Answers

Below, we provide the correct answers for the short exam questions.

- **Question 1**
  - **Question 1.1**: No.
  - **Question 1.2 (example)**: As statement coverage only asks for a certain line to be executed, only one test case suffices.
- **Question 2**: Two.
- **Question 3**: 2, because the condition must be covered for cases True and False.
- **Question 4**: $\{2, 3, 4\}$