Project

November 15th, 2024

Ana Paiva, José Campos

# 1. High-level goals

The high-level goals of this lab project are (1) to learn about systematic unit testing and (2) to reason about test quality, using different criteria, i.e., code coverage, mutation score, and smelliness of a developed test suite.

# 2. Setup

⚠️ Please make sure your machine is configured properly, i.e., Java JDK 21, Apache Maven v3.9.9, and that you have the latest version of IntelliJ IDEA installed. If not, please visit the [0. Setup](https://docs.google.com/document/u/1/d/1C0gMDh8pX77bOWW3Nypho5YMv4Z9gq7Lfis9R3Dt2eM/edit) document.

# 3. Subject

[**Bad Ice Cream**](https://drive.google.com/file/d/1LZIKbU8nlJRruY5Jxf0gzznStVA2GR4Q/view?usp=sharing). In this arcade game, you can help Bad Ice Cream steal his favourite fruits. But be careful, the fruit has an owner, and the monsters that protect it don't like thieves at all. Use Bad Ice Cream's power to create and destroy ice walls to trap monsters and fill your belly at will.

## 3.1 Instructions on how to run each subject

1. **Download** and **extract** the provided zip file.

2. **Open** it one IntelliJ.

3. **Run** the correspondent main function:

- The main function of the **Bad Ice Cream** subject lives in badIceCream.Game.

# 4. Assignment

In a nutshell, you must develop a comprehensive test suite for one of the Java-based retro games listed in Section 3.

For the subject assigned to you (see table below), you must:

1. **Download** and **extract** the provided zip file.

2. **Open** it one IntelliJ.

3. **Apply any testing technique/tool** to \*all\* Java classes in the assigned subject and develop a comprehensive test suite of unit test cases with [JUnit 5](https://junit.org/junit5/). You might use other Java libraries, e.g., [Mockito](https://site.mockito.org/), [AssertJ](https://assertj.github.io/doc/), or automatic unit test generation tools, such as [EvoSuite](https://github.com/EvoSuite/evosuite/) or [Randoop](https://github.com/randoop/randoop), or even [ChatGPT](https://chatgpt.com/) / [GitHub Copilot](https://github.com/features/copilot).

🧑‍💻Tip on how to compute the code coverage of the developed test suite.

// Run the following comand on the root directory of the project  
mvn test jacoco:report

// And then open the following file on your favorite browser

target/site/jacoco/index.html



🧑‍💻Tip on how to compute the mutation score of the developed test suite.

// Run the following comand on the root directory of the project  
mvn test-compile org.pitest:pitest-maven:mutationCoverage

// And then open the following file on your favorite browser

target/pit-reports/index.html



🧑‍💻Tip on how to compute the smelliness of the developed test suite. (You can find the .jar file of the [TestSmellDetector](https://github.com/jose/TestSmellDetector) tool [here](https://drive.google.com/file/d/1feLGWw8Dlg3kamS7bInM-N7aJsyxCVPv/view?usp=sharing).)

// 1.

// Create a tsdetector-input.csv file in the root directory of the

// project under test, e.g.,  
touch tsdetector-input.csv

// 2.

// Populate it with project's data (required to run [TestSmellDetector](https://github.com/jose/TestSmellDetector)).  
// For each test suite you develop, you must write a new row in the

// tsdetector-input.csv file. Each row follows the following format:  
// <project name>,<full path of the java file with the test suite>,<full path of the Java file of the correspondent class under test>

// For instance, lets suppose your project's name is XPTO and that you

// have a class under test that lives in

// /home/jose/XPTO/src/main/java/pt/up/fe/Foo.java

// and a test class (for Foo) that lives in

// /home/jose/XPTO/src/test/java/pt/up/fe/FooTest.java

//

// you should populate the tsdetector-input.csv as follows:

// XPTO,/home/jose/XPTO/src/main/java/pt/up/fe/Foo.java,/home/jose/XPTO/src/test/java/pt/up/fe/FooTest.java

// 3.

// Run [TestSmellDetector](https://github.com/jose/TestSmellDetector)

java -jar TestSmellDetector-0.1-jar-with-dependencies.jar \

--file tsdetector-input.csv \

--thresholds spadini \

--granularity boolean \

--output tsdetector-output.csv

// which produces a message like

Smelliness: 76.6%

// and an output file named tsdetector-output.csv which lists, per test

// suite, the set of smells present in each test suite.



# 5. Deliverables (deadline January 6th, 2025, 11:59:00 pm)

Each student should:

1. Zip the `tsdetector-input.csv` file, the pom.xml file, and the src/test directory. To do so, you might run the following command on the root of your project directory (⚠️replace <student-id> with your own student number):

zip -r <student-id>.zip tsdetector-input.csv pom.xml src/test



2. Upload the zip file to Moodle ([M.EIC’s link](https://moodle2425.up.pt/mod/assign/view.php?id=102873) / [MESW’s link](https://moodle2425.up.pt/mod/assign/view.php?id=103432)).

# 6. Academic integrity

As future IT professionals, we expect from students an irreproachable attitude, in both ethical and moral terms, as regulated by the [University policy on academic integrity](https://www.up.pt/portal/documents/8/codigo-etico-de-conduta-academica-uporto.pdf). Students may directly contact the [U.Porto's Ethics Committee](https://www.up.pt/portal/en/explore/organisation/ethics-committee/) for more information on this.

## Policy

**This project is designed to be done by you and you only**. We expect that students would discuss or talk about their projects with one another, but also expect that students are honest about their work. We expect each one works independently from one another, not exchanging source code or results with others. This implies not taking credit for others' work and not covering for others who have not contributed to the well-being of this course.

**Note that we will be using automated systems to detect source code plagiarism!**

Students may not copy any part of a solution to a problem that was written by another student, or was developed together with another student, or was copied from an unauthorized source. Students may not look at another student’s solution, even if they have completed their own, nor may they knowingly give their solution to another student or leave their solution where another student can see it.

Here are some examples of behavior that are inappropriate:

- Copying or retyping, or referring to, files or parts of files (such as source code, written text, or unit tests) from another person or source (whether in final or draft form, regardless of the permissions set on the associated files) while producing your own. This is true even if your version includes minor modifications such as style or variable name changes or minor logic modifications.

- Writing, using, or submitting a program that attempts to alter or erase grading information or otherwise compromise security of course resources.

- Lying to the course’s staff.

- Giving copies of work to others, or allowing someone else to copy or refer to your code or written assignment to produce their own, either in draft or final form. This includes making your work publicly available in a way that other students (current or future) can access your solutions, even if others’ access

is accidental or incidental to your goals.

- Coaching others step-by-step without them understanding your help.

It is not considered cheating to clarify vague points in this lab sheet; to give help or receive help in using the computer systems, compilers, debuggers, profilers, or other software; or even to discuss ideas at a very high level, without referring to or producing code.

Any violation of this policy is considered a fraud. **The minimum penalty for students, or groups of students, caught in a fraud such as plagiarism (either as plagiarists and plagiarized) is a 0 (zero) grade in this project.** Such incidents will also be reported through University channels, with possible additional disciplinary actions (see the above-linked University policy on academic integrity).

Students should feel free to reach out to the course's staff for any question or clarification about this policy.

# 7. Evaluation

The grade of any developed test suite will be computed as  
  
 65% x branch coverageꭓ + 35% x mutation score𝛼 - 10% x smellinessᵝ

We will assess branch coverage with the [JaCoCo](https://www.eclemma.org/jacoco/) library, mutation score with the [Pitest](https://pitest.org/) library, and smelliness with the [TestSmellDetector](https://github.com/jose/TestSmellDetector) tool.

ꭓ As the project under test might have dead code or code impossible to reach/exercise, we will compute the maximum achievable branch coverage by (1) building a super-gigant test suite with all our test cases + all test cases developed by all students and (2) running branch coverage. The branch coverage achieved by the super-gigant test suite will work as the maximum achievable branch coverage. For example, if the maximum achievable branch coverage is 86% for project A and your test suite covers 86% of the A’s branches, you will get 100% in the branch coverage component.

𝛼 As it is impossible to automatically detect whether a mutant is or is not equivalent, we will make an educated guess by (1) building a super-gigant test suite with all our test cases + all test cases developed by all students, (2) running mutation analysis, and (3) identifying the mutants that are not killed by the super-gigant test suite. The non-killed mutants will be considered equivalent and will not be taken into account to assess the mutation score of your test suite.  
  
ᵝ To compute smelliness we first compute the ratio of test suites affected by each test smell and then the average % of test suites affected by all smells. Formally, given S as the set of smells and T as the set of test suites, smelliness is computed as:

smelliness(T, S) =

Please find below the list of test smells we aim to consider and that [TestSmellDetector](https://github.com/jose/TestSmellDetector) supports. You can find the description and an example of each test smell on the tool’s [webpage](https://testsmells.org/) or in lecture #8’s slides.  
- Assertion Roulette (AR)  
- Conditional Test Logic (CTL)  
- Constructor Initialization (CI)  
- Default Test (DT)  
- Duplicate Assert (DA)  
- Eager Test (ET)  
- Empty Test (EmT)  
- Exception Handling (EH)  
- General Fixture (GF)  
- Ignored Test (IgT)  
- Lazy Test (LT)  
- Magic Number Test (MNT)  
- Mystery Guest (MG)  
- Redundant Assertion (RA)  
- Redundant Print (RP)  
- Resource Optimism (RO)  
- Sensitive Equality (SE)  
- Sleepy Test (ST)  
- Unknown Test (UT)  
- Verbose Test (VT)  
 ~~- Dependent Test~~