An Experience Report on Using FormalZ in a Class

Wishnu Prasetya
Utrecht University

Based on a study carried out during the course Software Testing & Verification at Utrecht University, the Netherlands. The course took place in the 4th quarter of the academic year 2018/19.

Students should learn writing formal specifications

In the world of fast churning software industry, we might wonder whether applying **formal methods** is a viable option, since formal proofs are arguably hard to produce. Even if we can get programmers with the needed mathematical skill to produce them, the process is too slow to keep up with the pace of modern agile development. On the bright side, we do not need to exercise the full scale of formal methods to reap its benefit. Much can already be gained just by writing formal specifications. This does not require sophisticated mathematical skills, and nowadays not even a separate specification language and a separate tool chain anymore. Many modern programming languages support lambda-expressions, which allows, for instance, predicate logic formulas to be expressed natively in the programming languages themselves.

Having formal specifications enables verification through automated testing or bounded model checking. While this benefit might be clear in the eyes of a computer scientist, convincing practitioners to write formal specifications is still not easy. The myth that any form of exercising formal methods requires sophisticated math remains, and this is perhaps also a message that programmers somehow picked up from their education, where lectures in formal methods tend to be terse and dry. Students receive more points from being able to construct formal proofs. Demonstrating the ability to write formal specifications receives less points, hence creating the perception that it is less important (whereas we just argued that it is a more usable skill).

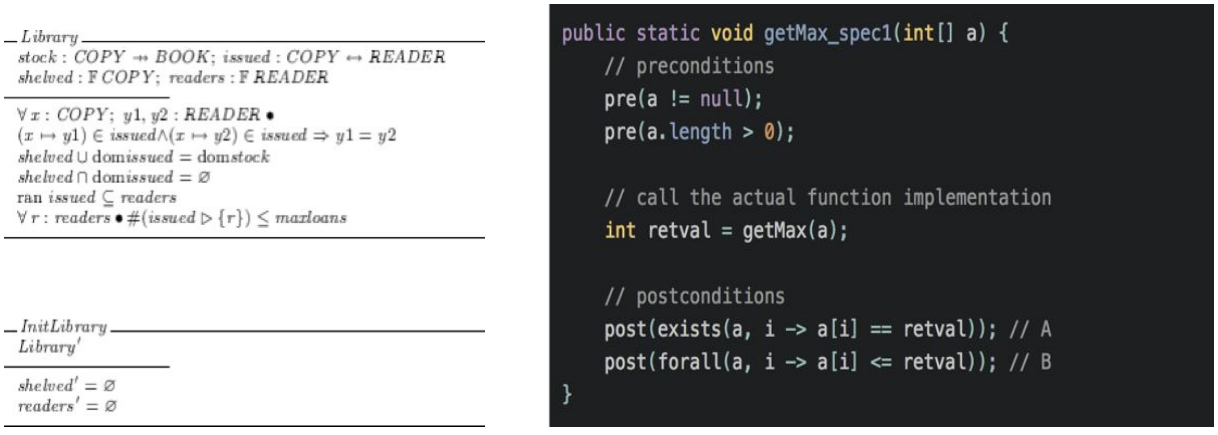

Figure 1. *Writing formal specifications used to require special mathematical notations that are hard to learn. However, nowadays modern programming languages have constructs that allow*

programmers to formalize specifications in the same programming language that they use to write programs.

The game FormalZ

FormalZ is a computer game that teachers can use to break the typical monotony in a class on formal method by allowing students to have some fun playing the game, while also learning the basic of writing formal specifications. Unlike typical software engineering themed education games, FormalZ takes a **deeper gamification** approach, where 'playing' is given a more central role. After all, what makes games so engaging is not merely the awarded scores and badges, but primarily the experience of playing them.

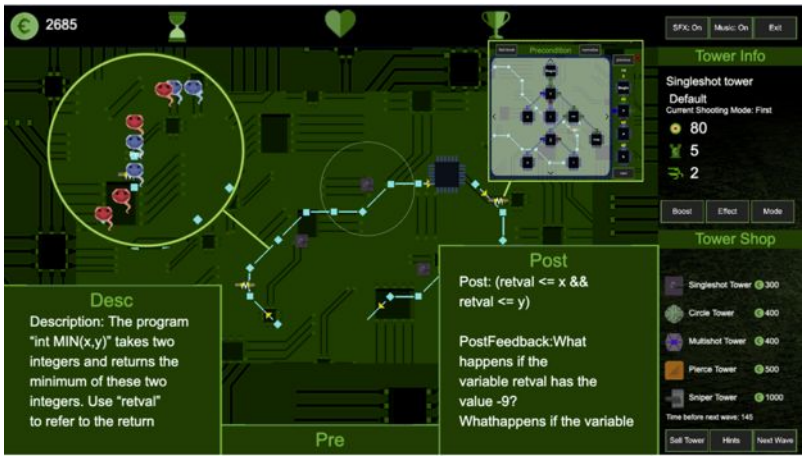

Figure 2: *a screenshot of the game FormalZ.*

In FormalZ, teachers provide exercises in the form of program headers and informal descriptions of the intended program behaviour. For each exercise, the student is asked to translate the informal description into formal pre- and post-conditions. An example of such an exercise is shown below:

> Given a non-empty array a, the program **getMax(int[] a)** returns the greatest element in the array.

A student solves the exercise by offering a pre- and post-condition which are equivalent to the teacher's solution. FormalZ frames this in a game of defending a CPU, which is a re-interpretation of the popular tower defence genre of games. The CPU symbolizes the program that is being specified. The story line is that hackers manage to find a way to influence data packages that flow into and out from the CPU. Data packages are represented by blobs, see Figure 2: red blobs are data that the hackers manage to corrupt to incorrect values, and blue blobs are those that are still 'clean'. No incoming red blob should reach the CPU, nor leave the CPU to reach the environment. To defend against this attack, the CPU's circuit board provides two scanners: one representing the program pre-condition, and one representing the program post-condition. Both can be programmed to identify and mark certain blobs.

To construct the pre- and post-conditions the student gets a special construction panel where blocks can be placed to construct a formula; see Figure 3. Each block represents either a variable, a constant or an operator. Wires are used to connect the blocks to construct the tree representation of the formula; its so-called Abstract Syntax Tree (AST). Simply typing the formula

is deliberately disallowed, as part of FormalZ' constructionism learning approach. The Constructionism theory of learning (originally proposed by Papert and Harel; Papert used it in his LOGO programming language to teach children to program). The theory believes that humans learn by constructing knowledge, rather than by simply transferring this knowledge from a teacher into the head of a watching learner. Familiar physical objects play a key role in this process, because the learner already has knowledge on how they work. When new knowledge is framed in terms of interactions with these familiar objects, it helps the learner to construct the new knowledge in his mind.

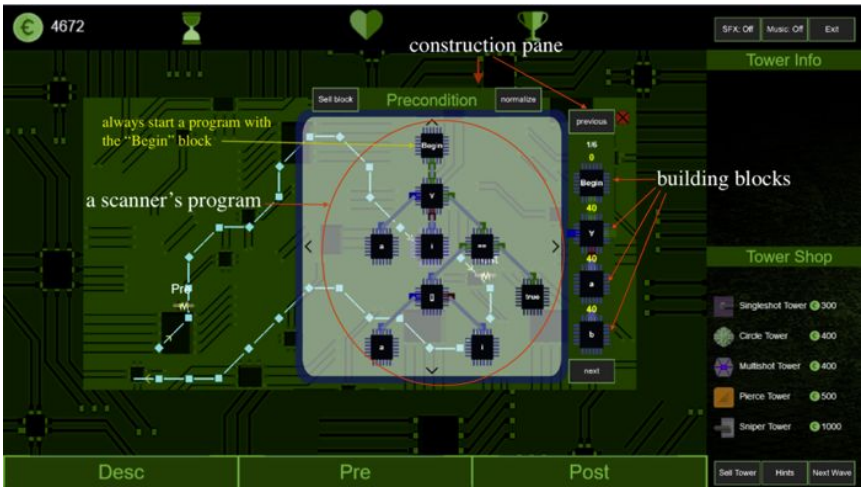

Figure 3: *FormalZ scanner construction pane.*

Studying FormalZ in a real class

To study the feasibility of using FormalZ in an actual class, we deployed FormalZ in the course Software Testing & Verification (STV) at Utrecht University. Embedding a game into a class is not as easy as it may sound. It will only work if it can be seamlessly embedded into the whole teaching and learning flow of the class. Since an education game like FormalZ is unique, there is no reference we can use to help us in designing the embedding. So, this study is also aimed at getting first feedback on how to do that.

Some basic information about the course is shown below. The course is aimed at introducing basic concepts of software testing, formal specification, and program verification to students. It is a bachelor course, which is also marked as an advanced level course at Utrecht University.

Course name:	Software Testing & Verification.
Level:	Bachelor, advanced level
Credit:	7.5 EC in 10 weeks
Teaching form:	2 x 2 hours lectures / week + 2 x 2 hours lab sessions / week
When:	23rd April - 28th June 2019
#Students:	80+

About 80 students registered to the course. Most are computer science students. There were a small number of students which were non-CS bachelor students were interested in doing some CS courses. CS Students had background in Logic, so formal method is not a new concept to them. However, this STV course is the first one where they really have to formalize the specifications of concrete programs (rather than just abstract examples in Logic textbooks).

FormalZ is deployed as an optional exercises to help students learning to write formal specifications. Developing an initial skill to write formal specification is one of the goals of the STV course, but note that it is not the only goal. So alongside FormalZ, there are other learning activities that take place to cover other goals. There are in total 5 FormalZ games of increasing difficulties. Each present a student with a program whose specification needs to be formalized. In total there were 19 students who did these optional exercises.

Overall it can be concluded that deploying FormalZ in a class of the size of STV is feasible, both in terms of computing and teaching resource. Students generally appreciated the idea of using a game like FormalZ to help them learning. However, integrating FormalZ to the STV course proved to be more challenging. The course is very dense. It has to cover plenty of advanced materials in just 10 weeks. Alongside FormalZ assignments, students also have graded home works and projects. This undermined FormalZ' role. FormalZ emphasized the fun element: students would learn better if they are more engaged,and fun improves engagement. However, we learned that just providing a game is not enough in order to create fun. If students are to have fun, they should also be given the opportunity/time to do so. This pre-condition was not sufficiently created during the course STV. This can be remedied this is by allocating more time to do FormalZ exercises, but to simply do this will be at the expense of other learning goals. So, an innovative way of embedding FormaZ without sacrificing existing learning goals is needed. This is future work.