GameCode: Choose your Own Problem Solving Path

Simon Liénardy Benoit Donnet

Université de Liège

simon.lienardy@uliege.be benoit.donnet@uliege.be

Motivations

- CS2 course that introduces a rigorous methodology to write programs using Loop Invariant [3], recursion, and basic data structures such as Files, Lists, Queues, and Stacks;
- COVID-19 lock-down forced us to switch to remote teaching. Instead of giving students yet another podcast in their course schedule, we gave them homework exercises, we called GameCode, that they could do at their own convenience.
- Exercises inspired by GameBooks in which the reader can choose the path they takes to complete the story. With GameCode, students can choose their own solving path for each exercise.
- Can be related to gamification [1, 2, 4, 5, 6]

Programming Methodology

Our programming methodology can be divided in four steps:

1. Introduce formal notations that will be helpful in the following;
2. Provide formal specifications (i.e. precondition and postcondition) of the problem;
3. Find a formal Loop Invariant (or a recursive formulation);
4. Build the code upon the Loop Invariant [3] (resp. the recursive formulation).

Algorithm 1 presents the links between the Loop Invariant and the code. This last can be derived from the specifications and the Loop Invariant: (i) from the *Precondition*, find the instructions `INIT` that lead to the Loop Invariant; (ii) from the Loop Invariant and the *Postcondition*, determine the condition under which the iteration must be stopped (i.e. $\neg$*loop_condition*), negate it to find the `loop_condition`; (iii) from the Loop Invariant and the `loop_condition`, derive the `LOOP BODY` instructions; (iv) from *Loop Invariant* $\wedge$ $\neg$*loop_condition*, find the `END` instructions that lead to the *Postcondition* and (v) show that the loop ends thank to a Loop Variant. The first four steps can be done in any order.

```
// Precondition
INIT
// Loop Invariant
while(loop_condition){
    // Loop Invariant ∧ loop_condition
    LOOP BODY
    // Loop Invariant
}
// Loop Invariant ∧ ¬loop_condition
END
// Postcondition
```

Algorithm 1: Links between code and Loop Invariant

GameCode

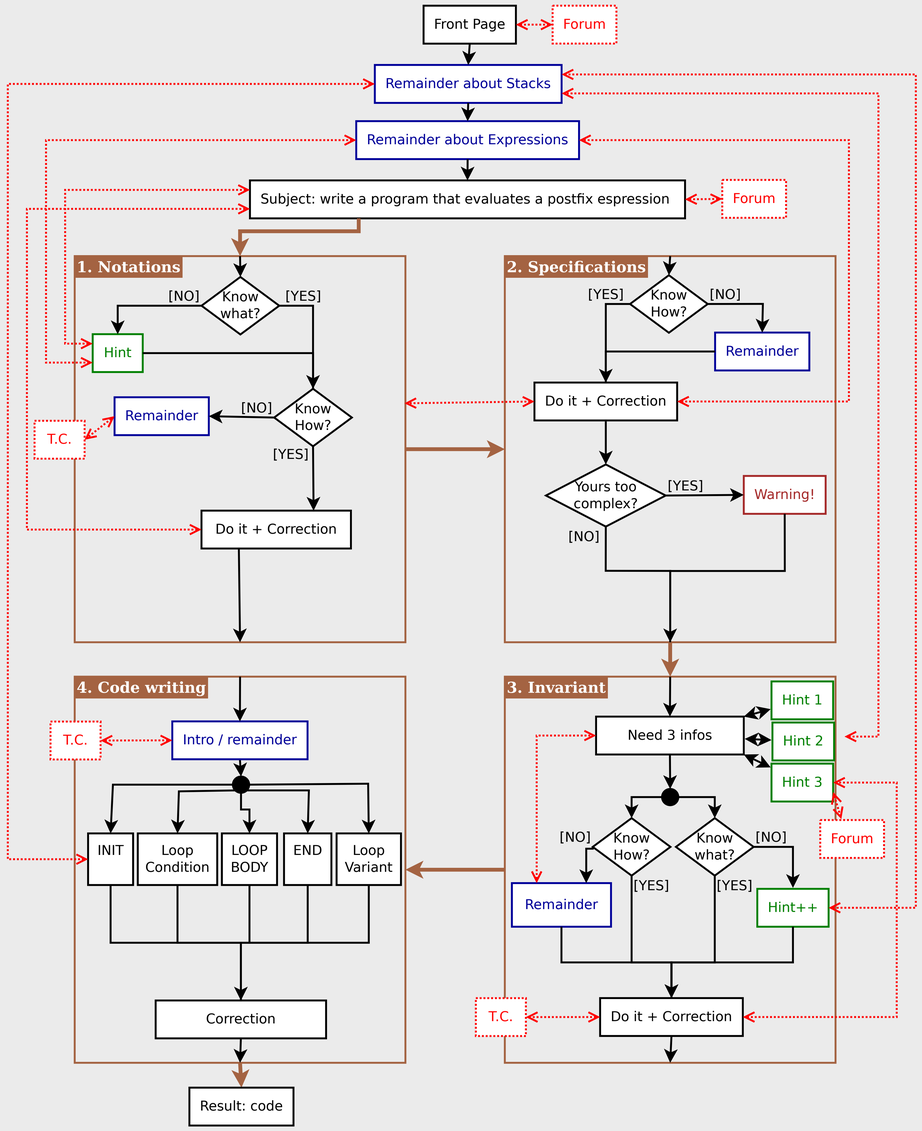

Figure 1: Example of GameCode map (with an exercise about Stacks and expressions). The four brown rectangles represent the main parts of our programming methodology and each of them should be read vertically. Plain arrows represent transitions available to the student. Dashed red arrows represent links to previous or remote content ("Forum" is the course forum and "T.C." stands for theoretical course. Any GameCode exercise meets the following requirements: (i) stand-alone book (a GameCode exercise is self-sufficient and contains the minimal information to complete the exercise), (ii) just-in-time theory (the theoretical reminders are placed where they are needed and are as short as possible), (iii) "no spoilers hints" (hints given to the students never reveal a solution, nor a part of it), and, (iv) no single solution (several solutions are always possible and a GameCode exercise always references the course forum to discuss them).

Preliminary Data

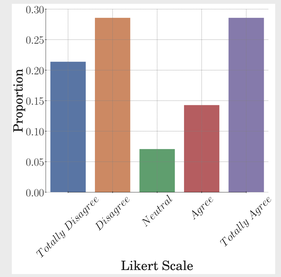

Figure 2: Survey: "I would have preferred to watch a podcast showing exercises resolutions." (N = 14)

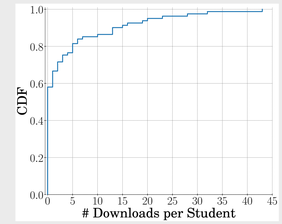

Figure 3: Number of GameCode exercises (14 available) download per student (N = 81).

- Few students took part in the exercises (See Fig. 3) (lower motivation due to COVID-19 lock-down + courses abandonment);
- From the Survey (Fig. 2), if we remove those who did not do the exercise, the numbers of agreeing (43%) and disagreeing (50%) students are close.

Future Work

We will address the following research questions:

RQ1 What are the typical paths followed by students in GameCode exercises? Answering this question should highlight the typical usage of GameCode by students and how they get to grips with the concept.

RQ2 How long does it take, to a student, to complete a GameCode exercise? Answering this question should provide clues on difficulties encountered by students, leading to a better assistance to students and an improvement in their productivity.

RQ3 Which step, in a GameCode exercise, appears to be the more difficult to students? This RQ is the natural follow-up of RQ2 and should refine conclusions drawn from RQ2.

RQ4 Which aspect of a GameCode exercise appears to be the most effective to students? This RQ will help to calibrate a GameCode exercise, in particular to focus on difficulties and students efficiency.

References

[1] Christo Dichev and Darina Dicheva. 2017. Gamifying education: what is known, what is believed and what remains uncertain: a critical review. *International journal of educational technology in higher education* 14, 1 (2017), 9.

[2] Darina Dicheva, Keith Irwin, and Christo Dichev. 2019. OneUp: Engaging Students in a Gamified Data Structures Course. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 386–392.

[3] Edsger. W. Dijkstra. 1976. *A Discipline of Programming*. Prentice-Hall, Inc.

[4] Brian Harrington and Ayaan Chaudhry. 2017. TrAcademic: improving participation and engagement in CS1/CS2 with gamified practicals. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. 347–352.

[5] Maria-Blanca Ibanez, Angela Di-Serio, and Carlos Delgado-Kloos. 2014. Gamification for engaging computer science students in learning activities: A case study. *IEEE Transactions on learning technologies* 7, 3 (2014), 291–301.

[6] Gina Sprint and Diane Cook. 2015. Enhancing the CS1 student experience with gamification. In *2015 IEEE Integrated STEM Education Conference*. IEEE, 94–99.