aMAZEing Programming — Providing SKILLs to Fellow Students

Timo P. Gros[1], Pascal L. Held[1], Pascal Lauer[1], Niklas O. Metzger[1], Kallistos Weis[1]

Abstract: Learning programming can be hard, especially for inexperienced freshman. An aMAZEing team of 30 students decided to lighten that challenge for their fellow students. This paper will present the attempt of volunteers to prepare their fellow students for their programming course, resulting in a week full of lectures, projects, coachings and didactical effort.

Keywords: students preparatory course; tutor preparatory workshop; programming; didactics; domain specific language

1 Introduction

Learning how to code is an obligatory task for every computer scientist. Starting without any programming skills is the reality for many students in early semesters. At Saarland University, two mandatory courses are supposed to teach the basics of programming where the focus is on theoretical aspects and the pace of the lecture and required exercises is very high. Therefore, starting four years ago, the first of our team members had the vision to improve the basic lecture “Programmierung 2”. Enjoying the trust of Prof. Sebastian Hack, the teaching professor, we started to organize a one-day students preparatory course (SPC) and beforehand a four-day tutor preparatory workshop (TPW). The latter was meant to prepare the tutors for their tutoring job and to improve their teaching ability. The SPC had the intention to teach the basics of Unix and Git, such that during the following semester the students can concentrate on the programming tasks without having the need to waste their cognitive abilities on technical prerequisites. We observed a significant increase in the number of students passing the course.

During the last four years, the team of current and former tutors kept improving the SPC. Still, many students were overwhelmed by the task of learning multiple programming languages within one course. Therefore, we decided to try to ease that challenge. We wanted to organize a one week preparatory course, teaching the students not a specific programming language, but especially *how to learn* a new programming language. That was when the aMAZEing Programming Course (APC) was born. For the APC we developed a domain specific language to solve mazes. We decided to not use an existing learn-to-program project as we wanted the project to suit our intentions exactly. The project we developed for the APC can be found on `https://amazeing.de`.

[1] Affiliations including {timopgros, s9paheld, s8palaue, s8nimetz, s9ksweis}@stud.uni-saarland.de

cc by sa

Study regulations According to the study regulations at Saarland University, in their first semester the students participate in the lecture “Programmierung 1”. This may raise the question, why a course as the APC is needed between the first and the second semester. The Programmierung 1 lecture deals with functional programming only [Sm08]. In opposite, the lecture of the second semester, called Programmierung 2, deals with imperative programming languages, such as C and Java. Even though the students already had a different — functional — programming course, our experience shows that students in Programmierung 2 are overwhelmed by learning multiple programming languages, especially at the beginning of the course. This is where the APC can help: teaching how to learn new programming languages and technical skills efficiently.

Statistics The APC took place one week before the start of the lectures, i.e. from Monday, April 1^{st} to Friday, April 5^{th}. There were 316 students participating. In comparison, the Programmierung 2 course has 540 participants. Considering that the course traditionally has many students reparticipating in the Programmierung 2 course and thus not visiting a preparatory course, we reached a large majority of the target students.

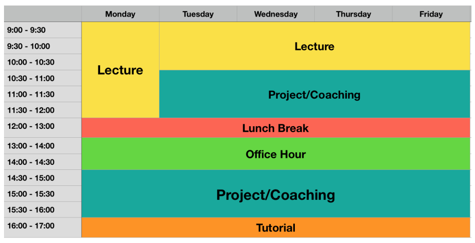
Our team consisted of 13 tutors, 11 coaches that mainly are alumni tutors and 6 supervisors. Of the latter, 2 were holding the lectures, 2 organized the aMAZEing project (AP) and 2 held the TPW.

Paper outline In the following, the organization of the aMAZEing Programming course is presented in section 2. section 3 and section 4 show the lectures and the aMAZEing project, respectively. In section 5 we provide details about the latest iteration of the tutor preparatory workshop. Afterwards, section 6 evaluates the first iteration of the APC and section 7 concludes this paper.

2 Organization

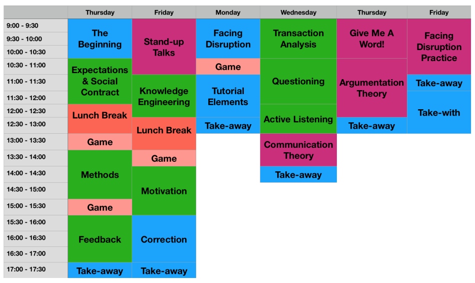
In this section, we present the organization of the APC. The timetable for the students and the APC team is shown as well as the content of tutorials and coachings.

As mentioned in section 1, aMAZEing Programming was a week-long course for the preparation of students before their second semester at Saarland University. The timetable of this event is pictured in Figure 1a. The four main parts that were relevant to the students are lectures, coachings, office hours and tutorials. In parallel to the APC, the tutor preparatory workshop took place but is not shown in this timetable. The schedule of the TPW is pictured in section 5. After the start of the day with the lecture, the daily project together with its documentation was published. In the first coaching phase, the students worked on the project by themselves, unassisted except for a small group of coaches that were supposed to help with technical challenges. After the lunch break, the office hour took place with tutors



	Monday	Tuesday	Wednesday	Thursday	Friday
9:00 - 9:30	Lecture	Lecture	Lecture	Lecture	Lecture
9:30 - 10:00					
10:00 - 10:30					
10:30 - 11:00		Project/Coaching	Project/Coaching	Project/Coaching	Project/Coaching
11:00 - 11:30					
11:30 - 12:00					
12:00 - 13:00	Lunch Break	Lunch Break	Lunch Break	Lunch Break	Lunch Break
13:00 - 14:00	Office Hour	Office Hour	Office Hour	Office Hour	Office Hour
14:00 - 14:30					
14:30 - 15:00	Project/Coaching	Project/Coaching	Project/Coaching	Project/Coaching	Project/Coaching
15:00 - 15:30					
15:30 - 16:00					
16:00 - 17:00	Tutorial	Tutorial	Tutorial	Tutorial	Tutorial

(a) aMAZEing Programming Course



	Thursday	Friday	Monday	Wednesday	Thursday	Friday
9:00 - 9:30	The Beginning	Stand-up Talks	Facing Disruption	Transaction Analysis	Give Me A Word!	Facing Disruption Practice
9:30 - 10:00						
10:00 - 10:30						
10:30 - 11:00	Expectations & Social Contract		Game	Questioning	Argumentation Theory	
11:00 - 11:30		Knowledge Engineering	Tutorial Elements			Take-away
11:30 - 12:00						Take-with
12:00 - 12:30	Lunch Break			Active Listening		
12:30 - 13:00		Lunch Break	Take-away		Take-away	
13:00 - 13:30	Game			Communication Theory		
13:30 - 14:00	Methods	Game				
14:00 - 14:30		Motivation		Take-away		
14:30 - 15:00						
15:00 - 15:30	Game					
15:30 - 16:00	Feedback	Correction				
16:00 - 16:30						
16:30 - 17:00						
17:00 - 17:30	Take-away	Take-away				

(b) Tutor preparatory workshop

Fig. 1: Timetables

actively supporting the students and answering questions. After a second coaching period, the solution to the project combined with additional exercises was presented in a tutorial.

To maximize the content as well as the knowledge of the participants, we split the day into multiple iterations of assisted and unassisted working phases. Therefore, the team of current and alumni Programmierung 2 tutors alternated in every time slot. In the following, we present the organization of coaching and tutorials, lectures are shown in section 3 and the tutor preparatory course follows in section 5.

2.1 Coaching

The coaching slot corresponds to the unassisted self-learning and -working phase for students. To prepare for the Programmierung 2 projects, each programming exercise was to be worked on by every student. However this does not mean that they were forced to work quietly on their own since discussing implementation ideas in small groups was always preceding the programming work. This helped to revisit their thoughts and potentially increase the quality of the ideas. We have noticed that the working phases led to friendships that we were still able to recognize throughout the Programmierung 2 course. Thus we are sure that our setting caused people to work together from the start in contrast to the observations made by Porter et al. [Po13]. Since technical issues were to be expected with the distribution of a virtual machine[2] running on multiple host systems, a group of up to 6 coaches was spread over all working areas for the students. They focused on fixing unintended software behaviour without helping to find a solution for the actual exercises. On top of that, they were always there to clarify parts of the assignment. Since "students have trouble following written instructions" this helped to avoid misunderstandings [Gi03]. During the coaching, the project team directly implemented and handled technical feedback. Therefore, the tutors in the office hours were prepared to help with common problems and answer questions.

[2] used to create a common starting point for the projects

2.2 Tutorial

In a project that is based on previous days, having a correct implementation before the next project release is necessary. Therefore, and for the convenience of the students, the end of the day was a tutorial in which the actual solution was presented. Furthermore, some independent exercises targeting common problems were treated. The tutor team that accompanies the Programmierung 2 course was also responsible for the APC tutorials. The new tutors were able to practice the knowledge learnt in the TPW and also get to know the students that will be present during the following semester. The content of the tutorials was prepared in team work as part of TPW which is dual to the mode of operation during Programmierung 2, where working as a team has a high priority.

3 Lectures

In this section, we present the concept of the aMAZEing Programming lectures. The structure as well as the topics provided by the lecturers is presented. We introduce the aMAZEing instruction language (AIL) as key contribution of this course.

The lectures of aMAZEing Programming are the start of every course day. The necessary background for each project day is explained to the participants. We provide a theoretical foundation as well as specific programming-language dependent information that enables to solve the projects and act as a linkage from aMAZEing Programming to the following course Programmierung 2. Since working synchronized on the servers provided by the university, knowledge of the given infrastructure is the starting point for the lectures. From day two on, the lectures are concerning Python, the aMAZEing instruction language and algorithms as well as programming concepts for the aMAZEing project. Every lecture ends with the presentation of the daily project with technical and conceptual information.

3.1 Unix & Git

Unix & Git are introduced on the first day of the course. Due to the amount of necessary commands that is inevitable for the efficient usage of a Unix system, the lecture of the first day lasts until the lunch break. The interactive lecture combines slides with information and on the fly execution of shell commands. Throughout the course, the poll system **DirectPoll**[3] works as interface between the lecturer and the students. Thereby, the audience's understanding can be checked and answering the questions provides short breaks that increase the attention. For a similar approach, namely Peer Instruction, there are observations that teaching with this technique clearly outperforms standard classes [Po13]. The slides are built to deliver a documentation that is complete w.r.t. the knowledge requirement of the course Programmierung 2 for Unix and Git.

[3] https://directpoll.com

3.2 Python

The participant's implementations that cannot be realised with the AIL in the front-end are written with the back-end language, i.e. Python. This highly used white-space sensitive language was chosen because of the readability and functionality of written code. In the lectures, syntax as well as explicit semantic definitions are presented with respect to the Python documentation [Py19]. On top of that, the main datastructures (i.e. lists, tuples, sets and dictionaries) and their advantages and disadvantages are part of the lecture, even though they are not (yet) part of the AIL. Since File I/O is a major topic in the Programmierung 2 course, it is also part of the aMAZEing project and introduced in the lectures combined with the concept of assumptions to validate input parameters. Thereby, a first glimpse for testing and correct software is presented to the students.

3.3 AIL: aMAZEing Instruction Language

The instruction language AIL is designed to suffice for implementing game-winning/maze-solving algorithms of the aMAZEing project. It is based on a stepwise execution of instructions defining the movement of a player in the labyrinth combined with conditional jumps to labels. The algorithms that are chosen to be implemented with AIL range from simple *Wall Following* to *Depth-First-Search* on the labyrinth. The AIL syntax is presented in Figure 2. Since the first project of the Programmierung 2 course has to be implemented in MIPS [He82], the structure of AIL is inspired by a low level machine language. To keep the focus on elaborating an algorithmic understanding AIL contains no sophisticated instructions. With this motivation, we prepare the students for the unintuitive low level language with a game based instruction language that abstracts from confusing arithmetic conventions.

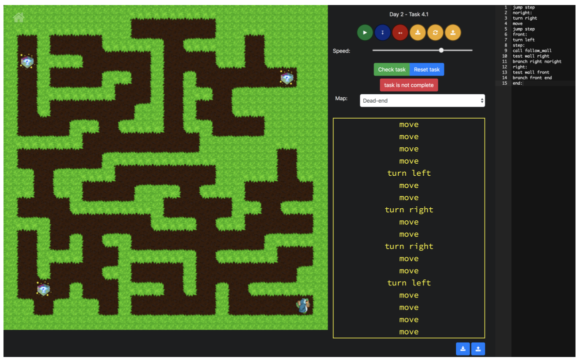
Statement	::=	`move` | `turn` *TurnDirection* | *Label*`:`
	|	`branch` *Label Label* | `store direction`
	|	`test direction` | `test wall` *PlayerDirection*
	|	`jump` *Label* | `call` *Label* | `return`
Label	::=	`[a-zA-Z]`$^+$
TurnDirection	::=	`right` | `left`
PlayerDirection	::=	*TurnDirection* | `front` | `back`

Fig. 2: Grammar of AIL

4 Projects

In this section we present the concept of the projects used in the APC. The tasks as well as the learning objectives for the students are presented.

We published the source code of the project on gitlab.com[4]. Figure 3 depicts the WebUI used for implementing tasks in AIL.



```
move
move
move
move
turn left
move
move
turn right
move
move
turn right
move
move
turn left
move
move
move
move
```

```
jump step
noright:
turn right
move
jump step
front:
turn left
step:
call follow_wall
test wall right
branch right noright
right:
test wall front
branch front end
end:
```

Fig. 3: Project WebUI of an AIL programming task.

The projects are the main part of every course day. The students experiment with the theoretical foundations provided by the lecture in a practical way to deepen the concepts mentioned in the lecture. As mentioned in section 3 the students work synchronized on the servers of the university such that the projects on the first day focus on the provided infrastructure. From day two on, the students work with one project that is described in more detail in subsection 4.2.

4.1 Git & Unix

Git & Unix are two short projects where the students practice the basics on how to work with projects under version control. The students explore the file system of Unix systems and how to create, copy and move files. They learn the most important things to work with Git.

4.2 aMAZEing Prog2

Motivation "The only way to learn a new programming language is by writing programs in it" quote by Dennis Ritchie. Most of the programming beginners do not know how to start their program. They fail to write the first statement because they do not try to solve the problem. Instead they think about how to solve the whole problem without doing it step by step. We try to encourage them by this project to solve problems step by step and do not fear to write and rewrite programs. "Playing games provides a visual representation to the abstract nature of the problem" [Ra05, p. 99]. This inspired us to provide a game like environment so students find it easier to come up with solutions to the given problems.

[4] `https://gitlab.com/amazeing/project`

Tasks To introduce the project to the students, they had to implement solutions for specific mazes in AIL as well as in Python by using the provided instructions and functions. In the next part, the students automate solving mazes by implementing algorithms in both languages, AIL (List. 1) and Python (List. 2). Figure 4 presents 3 exemplary algorithms whereby the last algorithm was not to be implemented in AIL. As one can see, the corresponding AIL implementation is really short and not hard to come up with. Similar exercises form an easy start and create positive first experiences. The successful start leads to a better performance throughout the remaining tasks [Wi05]. Furthermore the students had to implement a parser for mazes and had to check constraints to reject wrong input, i.e. wrong mazes.

- *Move until obstacle*: Move in a straight line as long as there is no obstacle (wall) in front of the player.
- *Wall following (right hand rule)*: Repeatedly follow the wall to the right of the player and turn right if wall ended and left otherwise until the goal is reached.
- *Depth-First-Search*: Nodes are the crossings in the maze and edges are the routes between the crossings.

Fig. 4: Selection of algorithms the students were ask to implement

```
step:
    test wall front
    branch end next
next:
    move
    jump step
end:
    return
```

List. 1: Move until obstacle in AIL.

```
from server.model import algorithms

def move_until():
    if algorithms.test_goal():
        return

    x, y = algorithms.get_player_position()
    x_off, y_off, name = algorithms.get_player_direction()

    while not algorithms.test_wall(x + x_off, y + y_off):
        algorithms.move()
        x, y = algorithms.get_player_position()
        if algorithms.test_goal():
            return
```

List. 2: Move until obstacle in Python.

We used web technologies as they are mostly platform independent and easily accessible. WebSockets gave us a reliable and simple two-way communication channel between front- and back-end. Instead of using raw WebSockets we used SocketIO which gave us an event based abstraction layer. The front-end code is standard HTML and ECMAScript 6 code. For the back-end we implemented the necessary routes (front-end delivery) using Flask and an SocketIO endpoint.

Used Resources We implemented the project using the following technologies and tools:

- WebSockets (using SocketIO[5]) [FM11]
- JavaScript (using yarn[6], Webpack Encore[7] and SocketIO[3])
- Python 3 (building on top of Flask[8] (-SocketIO[9]) and gevent[10]) [Py19]

5 Tutor Preparatory Workshop

One of the most important aspects of a lecture is its tutorials. The tutorials can make the difference between a really good and a really bad lecture. Thus, it is not only important to find good and motivated tutors, but also to prepare them as good as possible, especially if its their first time giving tutorials.

Simultaneously to the first SPC four years ago, the first TPW took place. The intention of creating such a workshop was for one thing to lower the time needed for familiarization of the tutors, such that they can start giving their very best with the first week of the semester. Furthermore, we observed through the years and several basic lectures, that the assignment to different tutors can have a major impact on the chance of passing the course. Thus, it also was intended to bring all the tutors to a similar state of knowledge regarding teaching, decreasing that impact as good as possible. On top of that, team building within the team of tutors was another important, if not even the most important, consideration.

The workshop consists of a mix of educational science, rhetorical exercises and best practice of tutoring. The latter being accumulated by many tutors through the years of giving tutorials, the rhetorical exercises being collected from and with professional coaches and the educational science being provided by our team members who studied not pure computer science but how to teach it.

5.1 Timetable and Didactical Concept

In the last years, the workshop mostly was given within four days of full time work before having the students preparatory course on Friday. With the APC, the new concept of the SPC, there was the need to find a new structure for the tutor preparatory workshop.

The workshop traditionally starts with some basics that are needed for further concepts. Thus, it is really important to have some days with the tutors having the workshop only, such that they can concentrate on their training without needing to work for the APC. On

[5] `https://socket.io`
[6] `https://yarnpkg.com`
[7] `https://symfony.com/doc/current/frontend.html`
[8] `http://flask.pocoo.org`
[9] `https://flask-socketio.readthedocs.io/en/latest/`
[10] `http://www.gevent.org`

the other hand, the APC's new structure offers an unique chance: as the tutorials take place in the afternoon only (and the lectures, office hours and coachings are supervised by others), we have the opportunity to teach the tutors in the morning and let them practice in the afternoon.

To combine both, we decided to first have two days (Thursday and Friday of the week before the APC) of full time tutor preparation and additionally use the mornings from Monday to Friday during the APC.

The timetable is displayed in Figure 1b. Hereby the colors give us the categories of the modules: 🟩 green for educational science, 🟦 blue for best practice and 🟪 purple for rhetorical exercises. Notice that the first days mostly contain modules of educational science, steadily decreasing with the beginning of the third day. The number of modules of rhetorical exercises and best practice in opposite increase over the workshops days. Furthermore, you can find team building games between the modules, not only meant for building a team but also to clear the mind of the tutors.

All the modules were designed with respect to the principle of independent learning: the instructions and talks were kept as short as possible. Instead, the focus was on group discussions and practical exercises. Simultaneously to pupils it holds: what is discovered or elaborated by the tutors themselves is learned deeper and better as if the same fact was just presented to them.

5.2 Take away — Take with

As you can see in Figure 1b, every day is ended with a short module called Take-away. Every student writes down some concrete actions they would like to remember during their work as a tutor on a card in a specific color. The cards then are collected. On the last day of the TPW, the tutors sit together in groups of size 3 or 4 and group the actions that were handed in several times. Then a moderator chosen from the group leads the discussion to rank the maps from most important to less important. These 10 concrete actions that can help being a better tutor can be revisited during the semester. Furthermore, the ranking was placed on the wall of the tutors lab.

5.3 Summary

The tutor preparation workshop does not only form a working team within a really short period of time, but also prepares the tutors in both, in a theoretical and a practical kind, for the upcoming job. According to their feedback, they feel more comfortable and more secure giving their first tutorials. During the last four years of preparing our tutors, the evaluation grades of our tutorials have increased significantly.

6 Evaluation

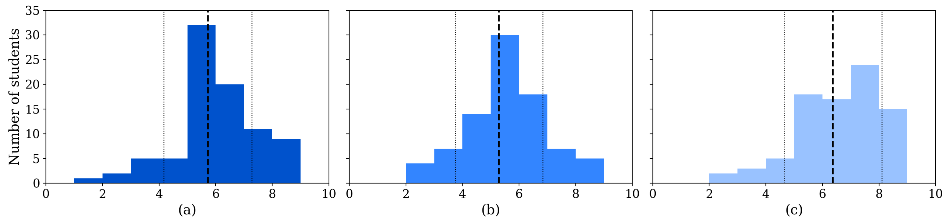


Fig. 5: (a) lecture speed; (b) lecture content; (c) project difficulty; dashed: mean; dotted: mean +/- standard deviation

As the summer semester has just started, there are no statistics available that could be presented to draw a conclusion whether the APC was successful with its intention to prepare our fellow students for their original programming course. Instead, we focus on internal and external feedback given by team members and students, respectively, and a preview of the APC's future.

We collected feedback by our students through a short form which was handed out and collected back in at the last day of the APC. We asked them to rate the pace of the lecture from too slow (1) to too fast (10), the content of the lecture from too less (1) to too much (10) and the difficulty of the project from too easy (1) to too hard (10). As the feedback of our students (see Figure 5) was mostly positive — the pace with average 5.72 and the content of the lecture with 5.29 were rated nearly perfectly — we plan to keep repeating the APC in a very similar way in upcoming iterations. The projects had an average value of 6.37, i.e. our students think, that the projects were a little bit too difficult. Thus, we plan some changes on the project.

6.1 Project

Our students rated the projects to be to difficult. We trace this back to the circumstance, that we used two different languages for the projects. The students first learned how to use AIL and later on had to use Python. While our intention initially was to show a real programming language, we actually do not see any didactical sense in conflicting them with more than one new language within one week. For future iterations, the AIL will be revised and extended, such that we not only can use this language, but also the language is exactly adapted to the content we would like to teach in an easily accessible and comprehensible way.

More on, we would like to revisit the AP itself. One major idea is to additionally introduce a multiplayer mode which can be used from students to play against each other. To do this without overcharging our team, we want to hand in the development of the project to the lecture "Software Engineering". This lecture gives the possibility to companies and chairs to submit projects. These projects will then be handled by a student team. We want to benefit from this chance to even improve the AP.

6.2 Lectures

In this years first instance of the APC "Testing" was not a part of the lecture. Still, it is an important topic of the following programming course. After the decision to not handle Python in future iterations and solely relying on AIL, we can reinforce our focus on testing in future iterations. The AIL language will be expanded in such manner that there will be a good way to teach testing and furthermore integrate testing into the AP.

6.3 Tutor Preparation Workshop

The concept of the TPW has had more iterations than the APC. Nevertheless, there is always the need to revisit the modules and keep them up to date.

While the concept itself is well known and well rehearsed, this year was the first time to enlarge the seminar, such that the tutors had the chance to practice their theoretical findings in the afternoon. As the tutors feedback in this regard was consistently positive, this format shall be kept for future iterations.

7 Conclusion

In this paper we presented the aMAZEing Programming course, an attempt to provide SKILLs to fellow students, preparing them for the programming courses at Saarland University and especially for Programmierung 2. We developed the aMAZEing instruction language, an easy domain specific language reduced to the fragments of programming we would like to teach and applied it to the aMAZEing project. The required knowledge was taught with especially developed lectures. To provide even further support, there were coachings and tutorials, the latter given by our tutors that were specially trained for this by the tutor preparation workshop. Even though it is hard to compare different iterations of Programmierung 2[11], one of the changes was so significant that it cannot be ignored. Namely the amount of feedback from students about our tutors' commitment. It seems like in this iteration students increasingly realized how many people are there to help them up if they fall down. By [Wi05] we conclude that this led to a boost of their self-efficacy and so performance in the course overall.

[11] The required coursework in terms of projects changes from iteration to iteration.

Acknowledgement

We are grateful for the confidence that the chairs Compiler Design Lab of Prof. Sebastian Hack and Foundations of Artificial Intelligence of Prof. Jörg Hoffmann have shown us. Thank you for giving us the chance to independently organize and execute the APC.

The most important shall be emphasized at the very end: without our team and their voluntary commitment, none of this would have been possible. We want to thank S. Ariesanu, J. Baumeister, S. Ehses, C. Eimer, G. Geier, J. Görgen, P. Hennen, E. Hoffmann, C. Kapp, D. Kempter, L. Kirschner, J. Klesen, D. Luche, N. Medinger, H. Meerkamp, M. Meiers, S. Oswald, J. Petermann, J. Schmidt, C. Schwan, L. Timm, S. Wack, F. Walgenbach, J. Warken and D. Weber. You are truely aMAZEing.

References

[FM11] Fette, I.; Melnikov, A.: The WebSocket Protocol, RFC 6455, RFC Editor, Dec. 2011, URL: `http://www.rfc-editor.org/rfc/rfc6455.txt`.

[Gi03] Giguette, R.: Pre-games: games designed to introduce CS1 and CS2 programming assignments. In (Grissom, S.; Knox, D.; Joyce, D. T.; Dann, W., eds.): Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, 2003, Reno, Nevada, USA, February 19-23, 2003. ACM, pp. 288–292, 2003, ISBN: 1-58113-648-X, URL: `https://doi.org/10.1145/611892.611990`.

[He82] Hennessy, J. L.; Jouppi, N. P.; Przybylski, S. A.; Rowen, C.; Gross, T. R.; Baskett, F.; Gill, J.: MIPS: A microprocessor architecture. In (Fisher, J. A.; Tracz, W. J.; Hopkins, W. C., eds.): Proceedings of the 15th annual workshop on Microprogramming, MICRO 1982, Palo Alto, California, USA, October 5-7, 1982. ACM/IEEE, pp. 17–22, 1982, URL: `http://dl.acm.org/citation.cfm?id=800930`.

[Po13] Porter, L.; Guzdial, M.; Mcdowell, C.; Simon, B.: Success in Introductory Programming: What Works? Commun. ACM 56/, pp. 34–36, Aug. 2013.

[Py19] Python Software Foundation: Python 3.7.3 documentation, 2019, URL: `https://docs.python.org/3/`.

[Ra05] Rajaravivarma, R.: A games-based approach for teaching the introductory programming course. SIGCSE Bulletin 37/4, pp. 98–102, 2005, URL: `https://doi.org/10.1145/1113847.1113886`.

[Sm08] Smolka, G.: Programmierung - eine Einführung in die Informatik mit Standard ML. Oldenbourg, 2008, ISBN: 978-3-486-58601-5.

[Wi05] Wiedenbeck, S.: Factors affecting the success of non-majors in learning to program. In (Anderson, R. J.; Fincher, S.; Guzdial, M., eds.): International Computing Education Research Workshop 2005, ICER '05, Seattle, WA, USA, October 1-2, 2005. ACM, pp. 13–24, 2005, ISBN: 1-59593-043-4, URL: `https://doi.org/10.1145/1089786.1089788`.