Experiential Factors Supporting Pupils' Perceived Competence In Coding - An Evaluative Qualitative Content Analysis

David Haselberger
Faculty of Computer Science
University of Vienna
Vienna, Austria
david.haselberger@univie.ac.at

Renate Motschnig
Faculty of Computer Science
University of Vienna
Vienna, Austria
renate.motschnig@univie.ac.at

Oswald Comber
Faculty of Computer Science
University of Vienna
Vienna, Austria
oswald.comber@univie.ac.at

Hubert Mayer
Faculty of Computer Science
University of Vienna
Vienna, Austria
hubert.mayer@univie.ac.at

Matthias Hörbe
Faculty of Computer Science
University of Vienna
Vienna, Austria
matthias@hoerbe.at

***Abstract*— This Full Paper in the research to practice category explores pupils' motivation to learn programming by game development in the game development environment (GDE) Unity^{TM}. After getting to know block-based programming through controlling robots, 66 students with minimal computer literacy between 14 and 16 years of age got in touch with Unity in six Informatics courses with group sizes between 8 and 14 students each for approximately 40 course hours. In particular, we are interested in experiential factors that underlie students' perceived competence in coding in high school computer science. We conducted an evaluative qualitative content analysis of 21 semi-structured interviews among students in six high school Informatics courses. Self-determination theory (SDT) identifies competence, autonomy and relatedness as basic needs that underpin motivation. We evaluated in how far students expressed these needs in their reciprocal interviews. Further, we explored student statements to find out about factors influencing their perceived competence. We were specifically interested in students' future expectancy to get in contact with coding - which we attributed to the need for autonomy. According to our findings, most students were rather motivated to code in Unity. Yet, many did not express a future interest in (game) development. The majority of students expressed perceived competence in the programming tasks. A few students also explicitly mentioned relatedness as important learning factor. In line with SDT, relatedness appears to be important for students' ability to deal with frustration along the journey of learning to code. It is open to consideration in how far students' social environment – including student peers awareness of technology use in their current personal surroundings and possible future opportunities for getting in contact with coding in their lives – is deeply relevant to students' perceived competence in programming apart from coding tools, such as Scratch or Unity, and incentives, such as programming games.**

Keywords—experiential learning, game development, Unity, qualitative content analysis, self-determination theory

I. INTRODUCTION

Programming is a "21st century skill" shaping new learning settings and school curricula. Computer technology drastically changes industries and markets. Social interaction is frequently computer-mediated and increasingly computer-facilitated. Many teenagers in the western hemisphere heavily use mobile devices and computer applications for their social contacts and orientation in the world [1], [2]. As such, a solid understanding for the underlying principles of what composes these applications – the art of programming – appears worthwhile.

Many educational programming environments, most notably Scratch, Snap! or Blockly, exist to facilitate computational thinking and motivate students on their coding journey through accessibility and practicability. These block-based programming languages have visual appeal and allow for easy, intuitive tinkering. They are designed to provide a low-threshold entry to programming, to support students' autonomous experimenting – within the confines of the environment – and to give positive incentives due to complexity hiding. These programming environments centrally deal with core triggers of frustration when implementing ideas in programming [3], [4] by avoiding these to begin with. Instead, they are conceptualized for being fun and gratify a need for competence [5].

The transition from block-based to text-based, more abstract programming languages appears to pose difficulties for learners [6], [7], [8], [9].

Many studies emphasize that learning text-based programming languages is prone to frustration and phases of demotivation (e. g. [10]). Frequently, a bimodal distribution of student results appears to emerge in programming competence exams [11], [12], [13]. While it has been discussed whether such exams really evaluate what they are supposed to [14], conceptual understanding of programming is, in general, considered difficult (e. g. [15]). Indeed, one basic driver for employing a block-based programming approach is to approach the steep learning curve typically associated with learning programming [16], [17].

So why should students in high school still get in contact with complicated and supposedly unintuitive text-based programming environments? Is it not sufficient to be a user and let the programming to the experts?

Programming with industry-standard coding tools provides us with a humble grasp of the jobs of computer scientists and their responsibilities. By acquiring such knowledge learners are enabled to make informed decisions on trusting one organization or another based on a general understanding of the subject matter.

Furthermore, getting acquainted with high-level, text-based programming languages is cognitive training: Even if you won't be a programmer in the future, you still can benefit from the challenges coming from text-based coding such as formulating problems, performing abstraction and generalization, and modelling solutions.

To become a critical user of real-world applications, the contact to real-world tools and programming approaches appears vital. If the mental model of handling animation in computer graphics sticks to euphemisms like "change costume", it's easy to understand and fun, but vastly over-simplified. "Change costume" can't give an idea of what causes glitches in a modern computer interface. It hardly allows for critical understanding of the executed code [18], [19] .

Block-based educational coding environments provide excellent easy-entry opportunities to grasp algorithms. Yet, they do not allow personal customization of computer-based learning or work environments. They do not allow access to the machine and operating system of the actual machine in use (with some exceptions such as QuartzComposer [20]). More importantly, they don't necessarily support the understanding and critical reflection of application source code that is actually used on a daily basis - unless purposely didactically mediated [21].

For younger pupils we can argue: "However, due to the young age, fostering intrinsic motivation for programming probably is of higher importance than making them see a connection to programming in real-life [5:57]." But is Scratch the prioritized tool to learn about programming in ninth grade Informatics with 14 - 15 year old students? In Austria, the ninth grade is the last (and so far only) year in which high school students get in contact with Informatics through an official school course.

Differing from educational programming environments such as Scratch or Blockly, Unity^{tm} does not allow for intuitive tinkering around without concept knowledge. Unity is a state-of-the-art game development environment (GDE) used by professionals around the world. Scripts associated with game objects and scenes are written in C# per default.

In the *"learn to ProGrAME" project*, researchers, teachers and pupils collaborated to find out about important moments in learning programming through game development in Unity. An underlying thesis of the project was: the prospect of game development is motivational to students even when faced with hurdles. In fact, learning game development was found to be perceived as exciting before starting to code by students participating in the project within its first iteration in the school year 2017/18. Yet, we found that during and after coding experiences in the first project iteration, the distribution of student motivation was evenly distributed [22]. Similar to findings by Ruf et al. [5] and Weintrop and Wilensky [23], many students perceived Scratch as easier, but were delighted to work with a professional tool [24]. However, a few pupils also explicitly mentioned that game development is not interesting to them since they don't play computer games and thus see no returned value for themselves.

Optimal challenge in learning programming through game development may be preserved by scaffolded learning projects that help build concept knowledge step by step, and provide opportunities to arrive at various game outcomes. The *"learn to ProGrAME"* tutorial is an open educational resource that is publicly accessible on GitHub (*https://learn2programe.github.io/learn2proGrAME-Tutorial/*). It offers step-by-step descriptions of game projects in Unity integrating fundamental programming concepts. Introduced programming concepts build on each other and so, the tutorial leads to ever more complex game challenges.

For many pupils participating in the *"learn to ProGrAME" project* and engaging in developing with Unity was not voluntary. Most considered it difficult and complicated despite stating motivation and interest stemming from the prospect to develop games (compare [24]).

Interestingly however, by far the most self-chosen team projects students worked on after the lessons on game development were completed with excellent results. And these projects - initiated and developed by the most part in student self-direction - were rather sophisticated and complex. Eight teams continued in creating and adapting video games - most of them in Unity.

Considering the importance of understanding high-level programming in a technology-supported community and the experiences of students with minimal computer literacy in game development through the *"learn to ProGrAME" project*, the central research question of this paper can be formulated as follows:

RQ1: *What context factors can be discerned that sustain students' perceived competence in coding in high school computer science classrooms?*

This study describes an evaluative qualitative content analysis of reciprocal interviews between 40 ninth grade students in six Informatics groups in an Austrian Realgymnasium. Due to privacy restrictions of the project, only 21 interviews were analyzed.

At the beginning of the course, students had little to no computer literacy skills. In the summer term, after 20 course lessons of developing games in Unity of two hours per week students were invited to interview each other on their experiences when learning to code and their future expectancies of their contact to programming.

According to self-determination theory (SDT), motivation and competence are interrelated. Internalization and integration processes operate in the service of psychological need satisfaction [25:202]. "The theory specifies that the more fully people internalize regulations of culturally valued extrinsically motivated activities, the more the PLOC (perceived locus of causality, author's note) will be internal, and the more the people will experience autonomy in carrying out the behaviors. Regulations that are less well internalized will have a more external PLOC, and thus behaviors will be more halfhearted, or dutiful, and there will be more experience of conflict. Variations in the quality of action and experience follow from these differences in relative autonomy [25:182]." Internalization is pivotal to satisfy the psychological need for competence. Further, SDT highlights that people internalize information because it enables us to feel connected to others. Thus, the psychological need of relatedness is central to mobilizing internalization processes [25:183].

In the study described in this paper, student interviews were evaluated by three researchers in an evaluative qualitative content analysis [26:123] regarding students' expressed self-perceptions of self-regulation [27], [25] and competence [28]. Students' mentions of relatedness, as well as comments on the online tutorial and the teacher were collected. Further questions they asked each other were registered.

In the next section, related studies are discussed, before the scenario of learning programming that forms the situational context of the interviews is described in detail. In section three, the evaluative qualitative content analysis of students' interviews is presented. Thereafter, results are discussed. The conclusion summarizes main findings and points to future research and didactics directions.

II. RELATED WORK

So far, there has been a reasonable amount of research on the topic of motivation in programming tasks. Ruf et al. [5] evaluated the motivational effects of using the block-based Scratch or the text-based Karol programming language for introductory computer programming lessons according to SDT in two seventh grade classes, with 56 students in total, in an Austrian Realgymnasium.

Identified regulation was measured higher for students in the Karol class. This was interpreted as showing that this programming environment is perceived more relevant for "real-life" than Scratch, which on the other hand was more fun for students. "Since the students at their age are too young to have a real grasp of 'programming in real-life' it seems to be the case that the way they are using Scratch does not relate to something they expect programmers to do – or maybe they are assuming that 'playing' with Scratch is just that – playing – and therefore unrelated to 'real life' [5:56]." It appears, however, that program structure visualization in Scratch was a better learning aid for children than program flow as presented in Karol.

Interestingly, even though students from the Scratch class performed better in a competence exam after programming lessons, they did not perceive themselves as more competent than the other students [5].

In a thematic context analysis of student exit interviews after participating in a redesigned computer engineering course at University, Trenshaw et al. [29] found three main topics emerging in interview discussions: "team projects promote relatedness; relatedness provides space for competence building, and without relatedness and competence, motivation declines [29:1200]." In a similar vein, the authors of this paper found in a qualitative content analysis of 128 learning journal entries of students in the first iteration of the *"learn to ProGrAME" project* that students described their ability to help others as personal achievements and working together as pleasant and fun [24], [30], [31].

III. SCENARIO

The evaluative qualitative content analysis in this paper represents experiences in learning to code of 21 students in six ninth grade Informatics groups (66 students in total) in the school year of 2018/19 at an Austrian Realgymnasium. The six Informatics groups refer to three classes with different specializations that were split into two groups each. One class has its specialization in "ecology and environment", the second is an inclusive class with specializations in "ecology and environment" as well as "global development and society" and the third has a "music" specialization. While each group was unique due to the interplay of group members and the teacher, all groups shared a similar Informatics context - the same laptops, the same room, the same teacher.

In the first *"learn to ProGrAME" project* iteration, ninth grade students of the school year 2017/18 were introduced to text-based programming and game development in three steps: a warm-up phase with block-based programming of interactive activities and Sphero SPRK+ robots (https://www.sphero.com), before programming in C# in the console and finally digging into the Unity GDE by implementing a platformer-style game. Working with Unity, lessons usually started with a teacher presentation on the next steps in game development process. Afterwards, students implemented these steps by themselves with teacher support, if needed.

Students expressed three major obstacles to learning programming in this first iteration of the project:

1. Students perceived the Unity GDE as tremendously complex,
2. Students wished for videos alongside teacher presentations to follow along and
3. Students critizised the lack of time for exploring the game development tool [24].

These were considered in the didactical design of the second iteration.

Again, students in all six groups were introduced to algorithms by programming SPRK+ robots to draw shapes on the floor in one lesson of two hours and block-based programming activities in Scratch in two lessons of two hours each. Students then directly started developing in the Unity GDE.

The complexity of the Unity GDE and C# programming was tackled with a scaffolded tutorial design [32]. The *"learn to ProGrAME"* tutorial was significantly enhanced compared to the first iteration to allow for learning basic programming concepts directly in Unity. It now included animated gifs and simple text descriptions of programming concepts and development steps. Further, a YouTube-channel describing the complete development of a platformer-style game is accessible through the tutorial.

Students worked in Unity for 20 lessons of two hours per week. This is a significantly increased time-frame to the first iteration with 6 lessons on console programming and 10 lessons on Unity.

After a teacher presentation of the development environment, one entire course unit in each group was spent on error handling, dealing with bugs and on opportunities for getting support. A main focus hereby was set on the fundamental Informatics learning concept of de-bugging [33], [34]. Then, students started out in Unity - importing graphics, saving a project and starting a game. Subsequently, user interaction and the concept of variable was introduced through the first game project, *Donut Clicker* (see fig. 1). Lastly, an enhancement to the game discusses possibilities of the "Transform" component of game objects in Unity.

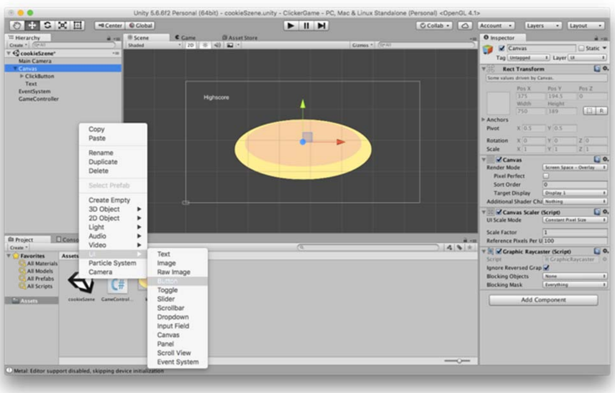

Figure 1: *Donut Clicker*

The next game project, an interactive graphic adventure game with text input, highlights the "string" data type, background graphics, and conditional statements (see fig. 2). This game project further introduces the concept of (game) states.

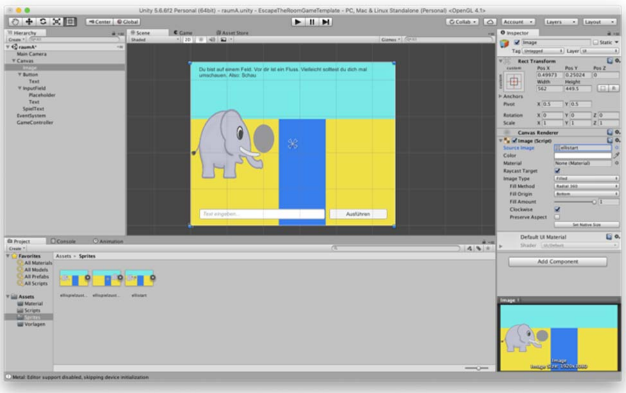
Unity 5.6.6f2 Personal (64bit) - raumA.unity - EscapeTheRoomGameTemplate - PC, Mac & Linux Standalone (Personal) <OpenGL 4.1>
Game
Asset Store
Du bist auf einem Feld. Vor dir ist ein Fluss. Vielleicht solltest du dich mal umschauen. Also: Schau
Ausführen
Add Component

Figure 2: *Elli's Adventure*

Following these beginner games, students in all six groups finished a "Space Asteroids" game that highlights the use of functions in C#.

Programming concepts were presented and discussed in teacher presentations, yet students frequently worked out programming tasks along the online tutorial. The teacher supported, but also delegated supporting to more advanced students.

In group discussions, students mentioned they liked the "Space Asteroids" game the most. It was the most "playable" game as much of the game logic, graphics and animations were prepared in advance. With minimal coding, students could play a fully featured arcade game. Many pupils also enjoyed implementing the "Donut Clicker" game. They highlighted that the development process was easy to understand. The text-adventure was least favored in development. In this game project, students had to code all the game logic - using concepts such as game states and conditionals - themselves.

After these three game projects, students worked on self-chosen team projects in Informatics within teams of 2 to 3 students for 10 lessons with 2 hours each weekly.

It was not prescribed to continue game development. Still, eight teams decided on implementing games in the course of their projects. Six teams chose Unity as GDE, one team used Scratch on a Raspberry Pi with buttons and LEDs, and one "team" (of one student by way of exception) chose to explore the Python scripting language. While two teams implemented the platformer-style game *BouncyFant*, instructed by the *"learn to ProGrAME"* learning resources, five teams developed creative adaptations beyond tutorial contents. In the end, a total of seven teams successfully implemented a complete *BouncyFant* game, an underwater platformer, a horror graphic text-adventure, a two-player 3d pong, a side-scroller jump'n'run with self-made graphics, a JSON-based text-adventure game engine and a "whack-a-mole" clone using electronics parts. Apart from game development, one team designed and constructed a wooden robot arm using a microcontroller - thus switching from C# to C. A team created a "magic mirror" using a Raspberry Pi and engaged in exploring Raspbian Linux. Another team conceptually designed a partner-matching algorithm whose student-moderated in-class evaluation led to an in-depth discussion on data privacy and beliefs in statistics. Another team implemented the game-of-life following a tutorial using VBA in Excel. A team prepared a complete lesson on the Dijkstra algorithm including student activities. Two teams hand-crafted encryption tools and worked out presentations on symmetrical and asymmetrical encryption techniques. Three teams taught themselves HTML and implemented webpages - for a band of friends, for the class, and for restaurant critics. Except for three, all self-chosen team projects in all six Informatics groups that followed mandatory programming in Unity were successfully completed.

IV. Method

A. Interview

To investigate the student's experience of learning how to code through game development in Unity, they were asked to conduct partner interviews with one another. These interviews were done after the learners had had gone through 20 course lessons in game development with Unity and finished two beginner and one advanced game project. Interviews took place before students started their self-chosen team projects.

Three interview questions were mandatory:

1. How was the programming experience for you?
2. What will you remember in five years?
3. How was the tutorial?

The first two questions hint at the students' perceived competence and motivation, while the third evaluates the quality of the learning materials. Moreover, students were asked to form at least one more question related to the game development sequences by themselves to ask their interview partner. These self-developed questions point at aspects of learning programming by game development that students themselves found important, necessary to explain or interesting to know about.

Students were allowed to choose for themselves how they wanted to conduct the interviews. They could choose what interview medium felt most comfortable to them. In total, 16 students handed in written interview accounts, 18 students produced audio-recordings of their interviews, and 6 students filmed their interviews. Table 1 shows distributions of handed-in interviews per Informatics group. In total, 40 interviews were handed in.

TABLE I. TABLE 1. INTERVIEW DEMOGRAPHICS PER INFORMATICS GROUP

Class	Group	Size	Interviews	Methods
“Ecology and Environment” specialization	A	10 (m: 3, f: 7)	4 (f: 4)	Audio (f: 4)
	B	11 (m: 11)	8 (m: 8)	Text (m: 2), Audio (m: 6)
“Ecology and Environment” and “Global Development and Society” specialization	A	15 (m: 9, f: 6)	10 (m: 8, f: 2)	Text (m: 2), Audio (m: 2, f: 2), Video (m: 4)
	B	8 (m: 2, f: 6)	8 (m: 2, f: 6)	Text (m: 2, f: 4), Audio (f: 2)
“Music” specialization	A	8 (m: 4, f: 4)	2 (f: 2)	Audio (f: 2)
	B	14 (m: 6, f:8)	8 (m: 2, f: 6)	Text (m:2, f: 4), Video (f: 2)

Written interviews ranged from approximately 100 to 350 words. Audio recordings were approximately 2 to 5 minutes in length, with two students responding 7 and 9 minutes respectively. Videos were approximately 2 to 5 minutes in length.

B. *Evaluative Qualitative Content Analysis*

As the classification and evaluation of contents according to constructs of a well-established theory, namely self-determination theory (SDT), are of central research interest in this study, interviews were analyzed in an evaluative qualitative content analysis [26:123].

First, categories were elaborated based on the SDT and an initial sighting of student interviews. Next, three independent coders met and discussed coding rules. Each coder individually went through the data case by case.

As all Informatics groups experienced lessons in the same room and with the same teacher, student interviews were not separately evaluated by class.

The units of analysis in individual cases were units of thought. Yet, as the evaluation of a specific text portion was frequently related to other text portions to be justified and thus depended on the surrounding text, coders decided on attributing whole cases to categories [26:43,140,141].

Apart from categorizing whole cases, we also gathered contributions not related to theory-guided pre-established categories.

Each case was coded independently by the three of the researchers. Of which two were not involved in teaching or grading Informatics groups. The third coder was the teacher of the classes. Pupils’ interviews were anonymized by one of the authors, who was the teacher.

After coding, we tested for interrater reliability of categorizations [35]. Agreement reliability was checked using Fleiss’ Kappa statistics. Typically, Fleiss’ Kappa values can be interpreted as such: Below 0 signifies poor interrater agreement, 0,01 to 0,2 slight agreement, 0,21 to 0,4 fair agreement, 0,41 to 0,6 moderate agreement, 0,61 to 0,8 substantial agreement and 0,81 to 1 almost perfect agreement.

The involved researchers met and discussed categorization discrepancies case by case to arrive at consensus on category attributions, especially for categories with initially poor to slight interrater agreements. Peculiarities were examined and classifications were finalized to arrive at a summative assessment of all cases [26:127,128].

We collected all contributions not related to any of the categories and summarized them according to the research question.

Quantitative statistics were correlated and compared to qualitative findings. Based on this cross-analysis, factors contributing to high school students’ sustained motivation were discerned and discussed.

Finally, a draft of the study was sent to two researchers not involved in teaching and coding, but participated in the *“learn to ProGrAME” project*, for member checking and study validation. Feedback on study design and interpretations were integrated.

C. *Category Definitions*

Coding categories were defined prior to the individual coding phase, mainly based on the SDT:

The first category is “Regulatory Style”. It is based on the organismic integration theory taxonomy of regulatory styles [25:193], [27]. We differentiated:

1. intrinsic motivation - interest, enjoyment, inherent satisfaction in learning,
2. awareness,
3. personal importance,
4. ego-involvement – as expressed in future endeavors (“Ego involvement illustrates that not all internal motivations are truly volitional or characterized by autonomy [25:181].”),
5. compliance - as expressed in references to grades or requirements, and
6. amotivation - non-valuing or disengagement.

In relation to the regulatory style, we were intrigued to find out more about the students’ “Future Expectations” of getting in contact with programming. We attributed this category to the psychological need for autonomy, that goes along the desire to be a causal agent of one's own life and act in harmony with one's integrated self. The category captures:

1. Future interest in game development and programming
2. No future interest because of self-image - expressed disinterest or indifference alongside self-valuation
3. No future interest because of game development and programming experience - expressed

disinterest or indifference alongside frustration in learning

Inferred from the "perceived competence scale" [28], the third category is "Perceived Competence". This category is divided into:

1. Expression of competence - confidence, capability, achievement, ability
2. Expression of frustration

Though not in the mandatory interview guideline, we collected students' notes on "Relatedness" and categorized:

1. Reference to relatedness - expressed in descriptions of interaction, social ties and connections, and caring for others

If not mentioned, we coded 0.

Dealing with the interview question on the "Learning Material" as mediator of learning experience, we coded whether students found it:

1. Understandable
2. Complicated

If not mentioned, we coded 0.

Further, we coded mentions on "Teacher Support" as mediating learning:

1. Supportive
2. Not supportive

If not mentioned, we coded 0.

For each case, each rater captured an anchor example from students' expressions along each categorization. Lastly, the answers to the self-defined questions by other students during peer interviews were summarized to gain insight into potential improvements for future project iterations.

V. FINDINGS

Table 2 lists summative content analysis findings including interrater-reliability test results for each category and exemplary anchor expressions within 21 student interviews.

TABLE II. EVALUATIVE QUALITATIVE CONTENT ANALYSIS FINDINGS

Category (Fleiss' Kappa; No. of Discussed Cases; Median; Average)	**Code No. - Label: Count (of 3*21=63 Classifications)**	**Exemplary Anchor Expression(s)**
Regulatory Style (0,63; 9; 3; 2,81)	1-Intrinsic Motivation: 11	"Very cool, I had interest in computer science all my life."
	2-Awareness: 17	"I found it really very interesting - and I had not much computer science knowledge"
	3-Personal Importance: 18	"Basically, I find programming not too bad"
	4-Ego-Involvement: 7	"For me it was ok. I could master the challenges most of the time. I sometimes needed help, but all in all I understood. So I think it was quite ok."
	5-Compliance: 10	"The programming experience for me was very challenging because it was not simple for me - specifically with C# - I had a huge struggle with variables, because I couldn't execute the variables, yes, that was it."
	6-Amotivation: 0	-
Future Expectations (0,62; 1; 2; 1,7)	1-Future Interest: 28	"To be honest, I don't know, but it is possible that I will get another thrive for programming"
	2-No Future Interest due to Self-Image: 23	"I don't know if this will help me in my future life, because I don't plan to do anything in this direction."
	3-No Future Interest due to Game Development Experience: 10*	"In 5 years I will probably remember that it was very challenging for me."
Perceived Competence (0,2; 10; 1; 1,4)	1-Expression of Competence: 37	"It was clear, what to do, to get to a good final product - and this was the most fun, to get to a cool end product - a game - after long thinking and exhausting working, and yes.“
	2-Expression of Frustration: 24*	"For me it was mainly that I didn't understand Unity, I didn't understand where I could find what. That was rather a negative experience."
Relatedness (1; 2; 0; 0,38)	1-Relatedness mentioned: 24	"Fun was when we were working together in a group of friends, and searched mistakes together and - worked together"
Learning Material (0,28; 0; 1; 1,4)	1-Understandable: 32	"It was quite well structured, and quite well explained. The smal GIFs and pictures were quite helpful."
	2-Complicated: 20**	"It was unfortunately partly rather unstructured. I oriented to pictures, because the text was not well described. It was not well detailed. You had to check in other things to look what you want. It was always described at the actual position - so a link to older stuff would have been helpful."
Teacher Support (0,33; 0; -; 0,22)	1-Supportive: 12	"Of course I needed sometimes help from the computer science teacher, but apart from that I could manage quite well on my own."

	2-Not Supportive: 1 ***	"Claims were made of things we had not learned."

* two interviews were not classified by one of the raters

** 11 categorizations as not relating to learning materials

*** 50 categorizations as no mention of teacher support

On average, students considered programming in Unity personally important. Several students were rather motivated (intrinsic motivation, awareness, personal importance, ego involvement). Only a few merely showed compliance to the demands of the course curriculum. No students showed amotivation towards the learning contents.

Regarding the category of future expectations, most students did not express a future interest in (game) development. However, as the results show, in majority this lack of interested stemmed from their self-perception as not interested in computers and having no need to learn programming rather than the experiences gained during the game development course. A lack of future interest due to the game development experience was occasionally mentioned in relation to a perceived difficulty of the tasks. Still yet, most students expressed they felt competent in the programming tasks, as to be seen from the perceived competence category.

Relatedness was frequently mentioned as important learning factor, despite not being prompted as response by the mandatory interview questions. This emphasizes the importance of relatedness in accordance with STD.

Learning resources and teacher support were considered supportive to large extents, if mentioned.

We collected the answers the students provided to the self-selected interview questions to gain further insight into their learning experiences. These were summarized by the two researchers not involved in teaching the class:

- The initial learning experience at the start of the project was perceived as especially taxing by two students.
- The fulfillment of programming goals was described by two students with feelings of pride and accomplishment.
- The act of building upon previous work in a continuous process with small achievements was described as exciting by one student.
- The prospect of programming being a useful skill for the future was directly mentioned by two students.
- One student noted that it is interesting to learn what lies behind video games and how they are constructed.
- The comparison of one's progress to the progress of other learners was mentioned three times by learners.
- Self-isolated learning while working on a tutorial document led to insecurities and demotivation for one student due to being unable to relate and compare learning experience with those of the other learners.
- Cooperation with other learners and growing together as a team was emphasized by four students as being fun and engaging aspects of the project.

VI. Discussion

Our results show that students tended to perceive game programming as important and exciting. Interestingly, they ascribed their disinterest in engaging in programming in the future to not seeing them going further in a computer science direction, rather than their experiences in starting out on learning programming.

From rater classifications, the following assumptions may be inferred:

A, There may be a link between students' regulatory style and perceived competence: Students that appear to be more intrinsically motivated tend to express more perceived competence and vice versa. In average, 10 students were found to be more intrinsically motivated (regulatory style < 4) and to express perceived competence (< 2) by all three raters (rater A: 10, rater B: 10, rater C: 11). Only 2 students, in average, are rather compliant (regulatory style > 3) and perceive themselves as competent (perceived competence < 2) according to the rater classifications (rater A: 2, rater B: 3, rater C: 1).

B, Students referencing learning together with others (relatedness = 1) appear to be more intrinsically motivated (regulatory style < 4) and rather tend to express perceived competence (< 2). In rater A's classifications, 5 of 7, in rater B's classifications, 4 of 7 and in rater C's classifications, 5 of 8 students mentioning learning together perceive themselves more intrinsically motivated and express more competence.

In line with SDT, pupils interested in programming games described themselves as rather capable in coding despite setbacks and difficulties in interviews. Moreover, students that see themselves getting in contact with code when they are older experienced themselves as capable programmers. In personal conversations with students that can see themselves involved in coding in their future lives, most explained that important peers, or family members, are working in the computer science field or are interested in it.

"The analysis of students' descriptions overwhelmingly pointed to relatedness as the most salient need in supporting their motivation in the course [29:1194]." Similar to Trenshaw et al.'s finding, in both iterations of the *"learn to ProGrAME" project* learning together with others in the group was perceived as supportive when dealing with learning hurdles [24], [31]. Relatedness and students' self-attributions and self-ideals appear to contribute significantly to students' motivation in learning programming.

The immediate and wider social environment of the learner, including peer groups and families, appear to be context factors that highly contribute to learners' perceptions of self-competence in programming. While no generalizations beyond the student groups are suggested, these assumptions are in line with findings concerning the SDT in other learning fields [25].

In the *"learn to ProGrAME" project*, the learning materials were designed for self-directed learning. However, to get the most out of the "one hour and fourty minutes"-lessons and to foster the students' learning, the classes consisted of a mix of (teacher-)lectures, teamwork and self-guided learning [22]. “While requiring the effective presence of the teacher - his or her orientation, stimulus, authority - that discipline must be built and adopted by the students [36:69].” The role of the teacher as an expert who is at least familiar with learning materials and students’ learning contexts, appeared important to a continuous progress of the game development activities. Providing support during the initial portion of programming lessons appears to be crucial, since several students mentioned the initial learning experience as taxing.

Students frequently commented on the dichotomy of fun and frustration derived from the programming task in the interviews. For many students motivation stemmed from the prospect of developing video games, which falls in line with previous research [22], [31].

When considering a transfer of the results of this study to undergraduate students our perspective is the following: Some of the results of this study, such as an increase in motivation to learn programming when connoted with activities that students tend to enjoy like gaming, would apply to students in higher grades and undergraduate students as well. Also, bridge-building courses for freshmen could directly be built upon the *“learn to ProGrAME”* curriculum. Nevertheless, academic curricula tend to target more solid and systematic programming skills such that they need to go beyond the *“learn to ProGrAME”* experience.

VII. LIMITATIONS

Researcher bias was handled by providing the study to two researchers that were not involved in the interview analysis process. Due to privacy restrictions of the research projects, not all 40 interviews that were handed in by students were included. Learning groups were from the same school and had computer science classes in the same room with the same teacher using the same tools and learning materials. However, sample size and research context can be seen as appropriate for an exploratory qualitative study design. In future research, we intend to evaluate our findings in different school settings of learning to code with Unity.

VIII. CONCLUSION

“Respect for popular knowledge, then, necessarily implies respect for cultural context. Educands’ concrete localization is the point of departure for the knowledge they create of the world. ‘Their’ world, in the last analysis, is the primary and inescapable face of the world itself [36:72].”

The evaluative content analysis of 21 reciprocal student interviews presented in this paper explores context factors that sustain students’ perceived competence in coding in high school computer science course environments based on SDT. Our findings suggest that the social environment in the classroom, and moreover in peer groups and families, contributes to students’ perceived competence in programming. Relatedness appears to be a core aspect of students’ ability to deal with frustrating obstacles in learning to code. Teacher’s expert support is important, particularly in the initial stages of starting out with coding. Developing games can be motivating, yet not all students are keen on computer games.

This suggests that the facilitation of a collective critical reflection of the necessity to learn programming and also the real-world use of computational tools in different areas of society is vital in computer science learning settings, even before selecting or designing the tools for specific educational purposes. Yet, interpersonal reconciliation in the classroom for young students to become critical, responsible participants in a technology-supported democracy needs time and, more importantly, the involvement of parents and the wider social school context.

ACKNOWLEDGMENT

The authors want to thank all students participating in this study for their engagement and sharing their insights.

REFERENCES

[1] M. Anderson and J. Jiang, “Teens, social media & technology 2018,” *Pew Research Center*, vol. 31, 2018.

[2] G. Mascheroni and K. Ólafsson, “Net children go mobile: Risks and opportunities,” 2014.

[3] D. Ford and C. Parnin, “Exploring causes of frustration for software developers,” presented at the 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering, 2015, pp. 115–116.

[4] R. C. Bryce, A. Cooley, A. Hansen and N. Hayrapetyan, "A one year empirical study of student programming bugs," *2010 IEEE Frontiers in Education Conference (FIE)*, Washington, DC, 2010, pp. F1G-1-F1G-7.

[5] A. Ruf, A. Mühling, and P. Hubwieser, “Scratch vs. Karel: impact on learning outcomes and motivation,” presented at the Proceedings of the 9th Workshop in Primary and Secondary Computing Education, 2014, pp. 50–59.

[6] M. Mladenović, I. Boljat, and Ž. Žanko, “Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level,” *Education and Information Technologies*, vol. 23, no. 4, pp. 1483–1500, 2018.

[7] P. Niemelä, “All rosy in scratch lessons: No bugs but guts with visual programming,” presented at the 2017 IEEE Frontiers in Education Conference (FIE), 2017, pp. 1–9.

[8] N. K. Simpkins, “I Scratch and Sense But Can I Program?: An Investigation of Learning with a Block Based Programming Language,” *International Journal of Information and Communication Technology Education (IJICTE)*, vol. 10, no. 3, pp. 87–116, 2014.

[9] A.-F. Lai and S.-M. Yang, “The learning effect of visualized programming learning on 6 th graders' problem solving and logical reasoning abilities,” presented at the 2011 International Conference on Electrical and Control Engineering, 2011, pp. 6940–6944.

[10] H. Tsukamoto, Y. Takemura, Y. Oomori, I. Ikeda, H. Nagumo, A. Monden, and K.-I. Matsumoto, “Textual vs. visual programming languages in programming education for primary schoolchildren,” presented at the 2016 IEEE Frontiers in Education Conference (FIE), 2016, pp. 1–7.

[11] C. Cabo, “Student Progress in Learning Computer Programming: Insights from Association Analysis,” presented at the 2019 IEEE Frontiers in Education Conference (FIE), 2019, pp. 1–8.

[12] L. J. Höök and A. Eckerdal, “On the bimodality in an introductory programming course: An analysis of student performance factors,” presented at the 2015 International Conference on Learning and Teaching in Computing and Engineering, 2015, pp. 79–86.

[13] A. Robins, “Learning edge momentum: A new account of outcomes in CS1,” *Computer Science Education*, vol. 20, no. 1, pp. 37–71, 2010.

[14] R. Lister, T. Clear, D. J. Bouvier, P. Carter, A. Eckerdal, J. Jacková, M. Lopez, R. McCartney, P. Robbins, and O. Seppälä, “Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer,” *ACM SIGCSE Bulletin*, vol. 41, no. 4, pp. 156–173, 2010.

[15] A. Gomes, F. B. Correia, and P. H. Abreu, “Types of assessing student-programming knowledge,” presented at the 2016 IEEE Frontiers in Education Conference (FIE), 2016, pp. 1–8.

[16] B. L. Santana and R. A. Bittencourt, “Increasing motivation of cs1 non-majors through an approach contextualized by games and media,” presented at the 2018 IEEE Frontiers in Education Conference (FIE), 2018, pp. 1–9.

[17] D. Weintrop and U. Wilensky, “Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs.,” presented at the ICER, 2015, vol. 15, pp. 101–110.

[18] K. Amanullah and T. Bell, “Analysing Students’ Scratch Programs and Addressing Issues using Elementary Patterns,” presented at the 2018 IEEE Frontiers in Education Conference (FIE), 2018, pp. 1–5.

[19] O. Ezenwoye, “What Language?-The Choice of an Introductory Programming Language,” presented at the 2018 IEEE Frontiers in Education Conference (FIE), 2018, pp. 1–8.

[20] Apple Inc. (2007). Quartz Composer User Guide. Retrieved from https://developer.apple.com/library/archive/documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide/qc_intro/qc_intro.html on 2020, Jan. 02

[21] K. Amanullah and T. Bell, “Evaluating the use of remixing in scratch projects based on repertoire, lines of code (loc), and elementary patterns,” presented at the 2019 IEEE Frontiers in Education Conference (FIE), 2019.

[22] O. Comber, R. Motschnig, H. Mayer, and D. Haselberger, “Engaging Students in Computer Science Education through Game Development with Unity,” presented at the 2019 IEEE Global Engineering Education Conference (EDUCON), 2019, pp. 199–205.

[23] D. Weintrop and U. Wilensky, “To block or not to block, that is the question: students' perceptions of blocks-based programming,” presented at the Proceedings of the 14th international conference on interaction design and children, 2015, pp. 199–208.

[24] D. Haselberger, O. Comber, and R. Motschnig, “Motivational Needs in Game Development using Unity in 9th Grade Informatics - A Qualitative Content Analysis“, edMedia 2020 (accepted).

[25] R. M. Ryan and E. L. Deci, *Self-determination theory: Basic psychological needs in motivation, development, and wellness*. Guilford Publications, 2018.

[26] U. Kuckartz, *Qualitative Content Analysis. Methods, Practice, Computer-Support (orig.: Qualitative Inhaltsanalyse. Methoden, Praxis, Computerunterstützung)*. Weinheim and Basel: Beltz Juventa, 2018.

[27] R. M. Ryan and J. P. Connell, “Perceived locus of causality and internalization: Examining reasons for acting in two domains.,” *Journal of personality and social psychology*, vol. 57, no. 5, p. 749, 1989.

[28] G. C. Williams and E. L. Deci, “Internalization of biopsychosocial values by medical students: a test of self-determination theory.,” *Journal of personality and social psychology*, vol. 70, no. 4, p. 767, 1996.

[29] K. F. Trenshaw, R. A. Revelo, K. A. Earl, and G. L. Herman, “Using self-determination theory principles to promote engineering students’ intrinsic motivation to learn,” *International Journal of Engineering Education*, vol. 32, no. 3, pp. 1194–1207, 2016.

[30] O. Comber, R. Motschnig, and H. Mayer, “Chat-Interviews as a Means to Explore Students’ Attitudes and Perceptions on Developing Video Games with Unity in Computer Science Classes,” presented at the International Conference on Interactive Collaborative Learning, 2019, pp. 903–914.

[31] O. Comber, R. Motschnig, H. Mayer, and M. Hörbe, “Best of Austria: Developing Video Games in Secondary Schools with Unity,” presented at the EdMedia+ Innovate Learning, 2019, pp. 181–189.

[32] E. C. Wenger and W. Snyder, “Communities of Practice: The Organizational Frontier ,” *Harvard Business Review*, vol. 78, pp. 139–145, 01-Jan-2000.

[33] A. A. Lowe, “Debugging: The Key to Unlocking the Mind of a Novice Programmer?,” presented at the 2019 IEEE Frontiers in Education Conference (FIE), 2019.

[34] S. Papert, *Mindstorms - Children, Computers and Powerful Ideas*. New York: Basic Books, 1993.

[35] J. L. Fleiss, “Measuring nominal scale agreement among many raters.,” *Psychological Bulletin*, vol. 76, no. 5, p. 378, 1971.

[36] P. Freire, *Pedagogy of Hope - Reliving Pedagogy of the Oppressed*. London: Continuum Publishing Company, 1992.