International Journal *of* Designs for Learning
2024 | Volume 15, Issue 1 | Pages 74-95

DESIGN AND DEVELOPMENT OF A DIGITAL PERSONALIZED LEARNING TRACK: BRIDGING THE GAP BETWEEN TEXTUAL AND VISUAL PROGRAMMING

Rani Van Schoors*[1,2], Sohum M. Bhatt*[1,3], Jan Elen[1,2], Annelies Raes[1,2], Wim Van den Noortgate[1,3] & Fien Depaepe[1,2]
[1]*itec–imec research group at KU Leuven;* [2]*Centre for Instructional Psychology and Technology at KU Leuven;*
[3]*Methodology of Educational Sciences Research Group at KU Leuven*

Due to swift technological changes in society, programming tasks are proliferating in formal and informal education around the globe. However, challenges arise regarding the acquisition of programming skills. Many students are unequipped to develop programming skills due to limited instruction or background and therefore feel insecure when encountering programming in higher education. Some after-school initiatives focus on teaching younger students programming skills, however, not all students have the opportunity to attend. It can also be very challenging for teachers to teach programming—even more so due to significant differences in students' knowledge and interests. To alleviate these challenges, we designed and developed a digital personalized learning (DPL) track for programming in the first grade of secondary education (12–14 year-old students) with a threefold purpose: (a) to encourage students bridging the gap between visual and more general-purpose textual programming languages (b) to meet differences in students' programming knowledge by challenging them, albeit on their own pace, and subsequently (c) to support teachers in the delivery of programming education with relevant supportive learning materials. The design was tested by students and teachers, both of varying technical abilities. Assessments of the DPL-track were positive, with students identifying the tasks as challenging and the tools as motivating. Teachers praised the adaptivity, as well as the gradual transition from visual to textual programming. We present several suggestions for design improvement and dilemmas while reflecting on our design case.

https://doi.org/10.14434/ijdl.v15i1.35224

Rani Van Schoors is a postdoctoral researcher in the itec–imec research group and the CIP&T research group at the Katholieke Universiteit (KU) Leuven. Interests include digital personalized learning and artificial intelligence in education.

Sohum M. Bhatt is a Ph.D. researcher in statistical modelling in the itec - imec research group and Methodology of Educational Sciences research group at the Katholieke Universiteit (KU) Leuven. Interests include user modeling and recommender systems.

Jan Elen is a full Professor in the CIP&T research group at the Katholieke Universiteit (KU) Leuven. Interests include the design of learning environments for complex learning, such as critical thinking.

Annelies Raes is an Assistant Professor in the CIP&T research group at the Katholieke Universiteit (KU) Leuven. Interests include innovative learning models, active learning and problem-based collaborative learning.

Wim Van den Noorgate is a full Professor in the itec–imec research group and Methodology of Educational Sciences research group at the Katholieke Universiteit (KU) Leuven. Interests include statistical modeling of clustered data using multilevel models, with a special focus on learning analytics and meta-analyses.

Fien Depaepe is a professor at itec–imec research group and CIP&T research group at the Katholieke Universiteit (KU) Leuven. Interests include instructional design and educational effectiveness of technology-enhanced learning environments.

*Rani Van Schoors and Sohum M. Bhatt are both considered first authors.

INTRODUCTION

The context of this design case is situated in the field of computational thinking and programming in education. We understand from the literature that, while the concept of computational thinking (CT) was first mentioned by Papert

(1980), it was Wing (2006) who (re)popularised it in education by describing CT most eloquently as a new basic skill alongside reading, writing, and arithmetic. She summarises CT as a process of thought involved in formulating problems and subsequent solutions to be effectively carried out by information-processing technology (Wing, 2006). Similarly, we noticed that programming—often regarded as the central practice of CT—also gained popularity in literature (Tedre & Denning, 2021). As Grover and Pea (2013, p. 40) state: "Programming is not only a fundamental skill of computer science and a key tool for supporting the cognitive tasks involved in CT but a demonstration of computational competencies as well." Programming gained currency in standards for primary and secondary education as well as a primary focus of EdTech companies and after-school learning initiatives (Falkner et al., 2015). Despite the ubiquity, various sources inform us that students' programming acquisition is hampered by three distinct difficulties that we aim to account for during the development and design of our case: (a) difficulty of content, (b) variation in student prior knowledge and (c) difficulty in teaching programming.

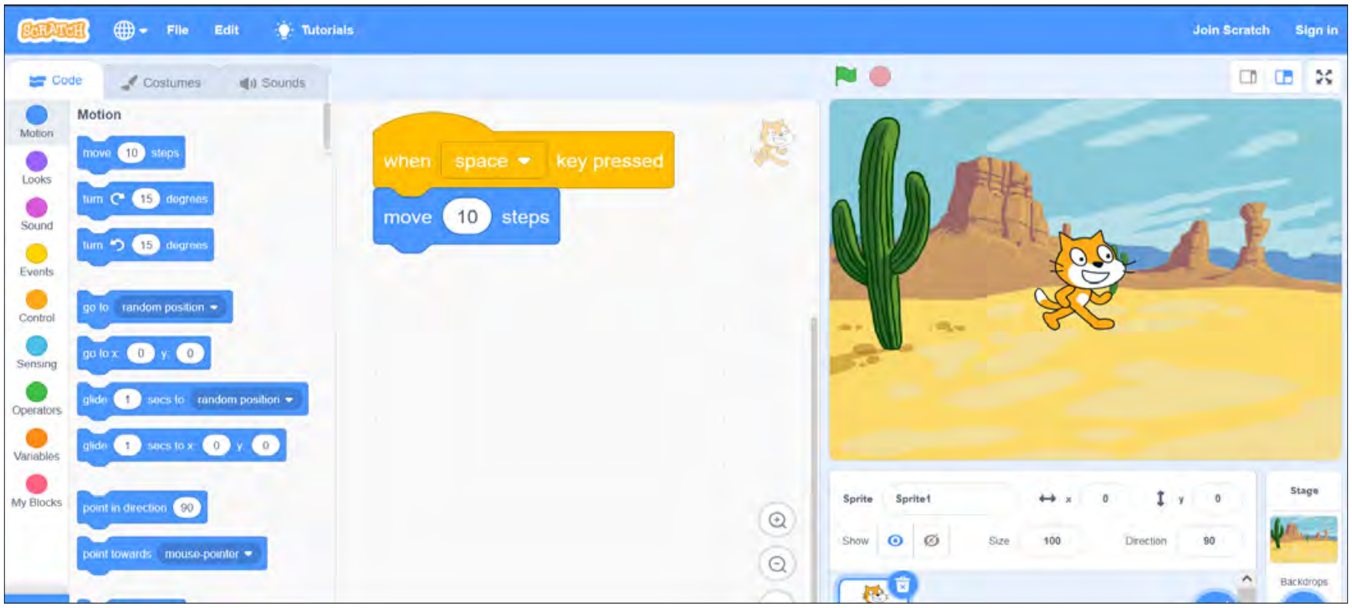
Driven by sources such as Govender (2006) and Jenkins (2004), we could distinguish a first difficulty that is endemic to computer science: learning how to program is difficult due to the multitude of skills and involved thought processes. Specifically textual programming is difficult to acquire as it requires syntax and logic to build programs. Kelleher and Pausch (2005) indicate that this increases students' cognitive demands. Despite the growth of programming courses for younger students, pass rates of introductory courses in higher education are still low (Bennedsen & Caspersen, 2019). Drawing upon our theoretical understanding, we believe that one plausible explanation is that students struggle to apply previously learned logic and simplified syntax (generally limited to visual programming via graphical elements) in other (textual) abstract syntax contexts (Saeli et al., 2011). Therefore, when building tools for programming, a design challenge emerges to include more adequate teaching approaches regarding introductory programming instruction, more specifically with special attention to ease the transition (less abrupt, more seamlessly) from visual to textual programming (Bennedsen & Caspersen, 2019; Bruce, 2018; Noone & Mooney, 2018).

The second difficulty that we distinguish is a substantial divide in students' programming knowledge and interests. Informal learning initiatives arise which aim to engage younger programming novices. Such endeavors enable interested students to get acquainted with programming outside of the classroom. However, as not all learners are able (or eager) to attend informal programming activities, a divide can be noticed in programming knowledge and interests between students (Salac et al., 2021). Therefore, when building tools for heterogeneous student groups, a design challenge emerges to facilitate the personalization of the learning experience according to students' needs. The field of digital personalized learning (DPL) proliferates and subsequently, many personalization types arise (Bernacki et al., 2021; FitzGerald et al., 2018). Based on this evidence in the literature, we pose an additional challenge: recognizing different personalization types and selecting the most fitting one for the target audience.

Finally, the preceding two difficulties coalesce into a third issue, which pertains to the difficulty of teaching programming. Teachers often feel distressed and unprepared because programming is an unfamiliar subject in their training (Brown et al., 2014). In recognizing this, we focus on an additional design challenge: when building tools for programming, it is important to create materials that support teachers to adequately teach programming. Also, as explained, it is important to consider that class contexts usually comprise students with different knowledge levels and interests, which challenges the teachers' role even more.

THE DESIGN OF THE DPL-TRACK AS AN ELEMENT OF A DBR-INITIATIVE

To alleviate the difficulties and accompanying design challenges, we pursued the design and development of a DPL-track (a personalized sequence of activities adapted to students' knowledge) for programming in the first grade of secondary education (12–14-year-old students). The context giving rise to this design endeavor is the research project (i-Learn) in which the design team explored the needs of teachers regarding digital personalized learning. As the previously mentioned difficulties (introduction) matched the teachers' needs, the design and development of the track -which is the foreground element of this article- unfolded. To do so, we took into principles of design-based research (DBR). McKenney and Reeves (2014) describe DBR as: "a genre of research in which the iterative development of solutions to practical and complex educational problems provides the setting for scientific inquiry" (p. 3). The development of DBR-solutions involves the aims of improving practice and contributing to theoretical understanding in close collaboration with corresponding stakeholders (McKenney & Reeves, 2012; Van den Akker, 1999). For this track, the design team consists of two researchers who worked together with software developers, teachers, and students. After developing the first prototype, a two-fold testing and evaluation cycle was set up. First, the DPL-track was first tested by seventeen students (12–14-year-old, 2 females and 15 males) and later evaluated through focus group discussions. Next, nine ICT/STEM teachers (four female and five male) evaluated the DPL-track and were likewise interviewed, albeit one-on-one. In terms of programming competencies, one teacher estimated himself as highly skilled, while four teachers rated themselves as moderately skilled, and four other teachers considered themselves to have low skills in programming. All evaluation moments served as solid

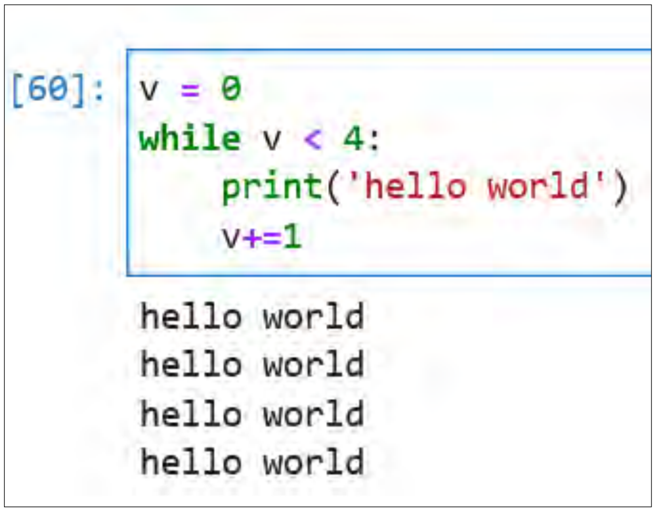


```
when space key pressed
move 10 steps
```

FIGURE 1. A view of Scratch; a visual programming environment in which students can produce small films, games, etc.

'co-learning' moments, as observations and dialogue gave insights into the needs and perceptions regarding the user experience and the design of the DPL-track.

Noteworthy, the expertise of the two main researchers is complementary: One researcher has a background in statistical modeling and is experienced in programming. The other has a background in educational technology and focuses on DPL. As programming and DPL are the twofold basis of the DPL-track, we will first elaborate on accompanying programming and DPL-theory which guided design choices concerning the prototype. Subsequently, we will elaborate on the design of the prototype and further elaborate on the development and refinement based on teachers' and students' comments.



```
[60]: v = 0
      while v < 4:
          print('hello world')
          v+=1

      hello world
      hello world
      hello world
      hello world
```

FIGURE 2. A view of Python; a textual programming environment.

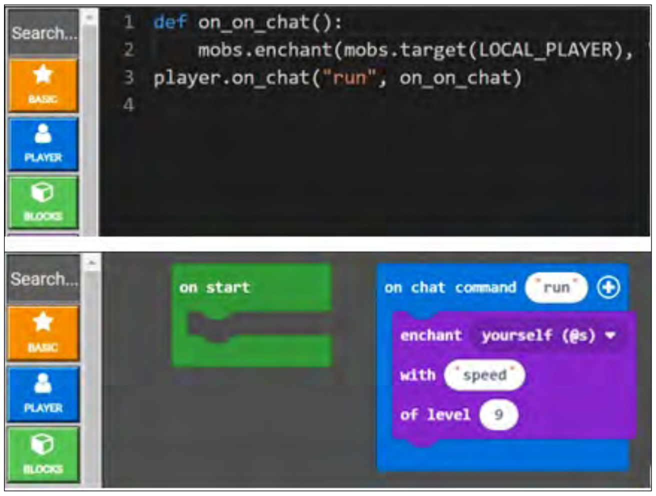
PROGRAMMING AND DPL IN EDUCATION

Programming

Myers (1990) describes the act of programming as "submitting a set of statements as a unit to a computer system to direct the behaviour of that system" (p. 98). How programming is introduced in the classroom is often through visual programming to obtain an understanding of the basic principles before learning textual programming. However, the transfer between textual and visual programming is a common struggle for students because of great discrepancies in syntax and semantics (Homer & Noble, 2017; Noone & Mooney, 2018; Tóth & Lovászová, 2018). To ease students in this transfer, hybrid programming can provide a middle ground. These three types of programming are further explained and depicted in more detail as they also appear in our DPL-track.

Visual programming

Visual programming operates via graphical elements, mostly symbols or icons, which represent codes and can be manipulated to build programs. Visual programming environments (e.g. Scratch, see Figure 1) are often used as an introduction for novice programmers as it is more inherently engaging and easier to understand than textual programming given the decreased cognitive demands and easier language rules (also known as syntax) dictating the structure of code combinations (Grover & Pea, 2013; Lye & Koh, 2014). The visuals allow for simpler debugging and testing as students can easily simulate the outcome of their code in the produced animation. Despite the benefits, visual programming languages have been relegated to teaching tools and are unsuitable for general programming in authentic computer systems (Lye & Koh, 2014).



```
def on_on_chat():
    mobs.enchant(mobs.target(LOCAL_PLAYER),
player.on_chat("run", on_on_chat)
```

FIGURE 3. A view of a hybrid programming environment for Python.

Textual programming

Unlike visual programming languages, textual programming languages are used generally and professionally (Chen et al., 2019). Instead of symbols or icons, written text is being used to build programs and applications (see Figure 2). Frequently used examples of textual programming languages are C, Java, and Python (Chen et al., 2019). Each one has its specific syntax which makes it difficult to learn for students. In addition, the textual and detailed nature of these programs involves more effort to understand and debug code (Lye & Koh, 2014).

Hybrid programming

Whereas visual programming can be a gentle introduction to programming, transitioning from visual to textual programming languages is a common struggle for students (Homer & Noble, 2017; Noone & Mooney, 2018; Tóth & Lovászová, 2018). To bridge this gap, hybrid programming has recently been developed (Noone & Mooney, 2018). These hybrid programming environments combine visual and textual programming: students can explore and edit visual constructs and simultaneously see reciprocal changes in the corresponding textual constructs (see Figure 3). In doing so, they learn textual languages' syntax, order of command executions, and other textual considerations (Robinson, 2016).

Programming in Flemish Education

In the setting of our design case, the government of Flanders (Belgium) announced new educational standards concerning CT and programming for the first grade (12-to-14-year-old students) in September 2019. The standards comprise two vague competencies: (a) Students can identify the foundations of digital systems and (b) students can apply a simple self-designed algorithm to solve a digital or non-digital problem. Often, teachers find it challenging to implement programming lessons into their curricula: many feel unprepared as CT and programming are not always fully included in their training (Sands et al., 2018). Nonetheless, by providing new educational standards, the government aims to integrate programming transversally, across multiple courses by teachers with different backgrounds (Flemish Government, 2019; Flemish Parliament, 2018). However, only a few Flemish schools have adopted programming into their curricula, mostly limited to very basic learning content in visual programming environments and taught by ICT teachers. In addition, teaching programming can be complicated by large heterogeneous classes, involving students with various knowledge levels and interests (Capovilla et al., 2015; Gomes et al., 2012). With all of this taken into consideration, a clear need arises for new learning tools, which can be tailored according to student's needs and are in line with the government's standards. To meet this, we integrated adaptivity within the DPL-track, based on DPL-characteristics.

DPL

In the last 25 years, much research has been done on DPL (Bernacki et al., 2021; FitzGerald et al., 2018; Major & Francis, 2020; Xie et al., 2019). We use the definition of DPL defined by Van Schoors et al. (2021, p. 14), who reviewed 53 manuscripts during that period and concluded on the following characteristics: "Unlike conventional learning, digital personalized learning takes place in a digital learning environment that adapts to the individual learner in the function of optimizing individual and/or collaborative learning processes focusing on cognitive, affective, motivational, metacognitive and/or efficiency outcomes. This adaptation/personalization: (a) can take into account cognitive, affective, motivational, and metacognitive characteristics of the learner; (b) can relate to all aspects of the learning environment, more specifically the (nature, number, and sequence of) learning tasks, the content as well as the instruction and support provided by the learning environment; (c) can be the result of information provided by the teacher or the learner himself/herself, but also information collected by the digital environment; and (d) can be enhanced by the teacher through the effective use of data derived from the digital personalized tools).".

Prior research found limited teacher awareness of DPL in Flanders, resulting in low use of DPL-tools. However, participants showed positive perceptions and an active willingness to implement DPL-tools in their future lessons. A clear need was detected for tools focusing on (relatively new) subjects that are technology-related. With this respect, the need for adaptive material to teach CT was frequently mentioned by participants (Van Schoors et al., 2023a).

All needs concerning programming and DPL bring us to the goal of this design case: the development of a DPL-track for CT in the first grade of secondary education. During the

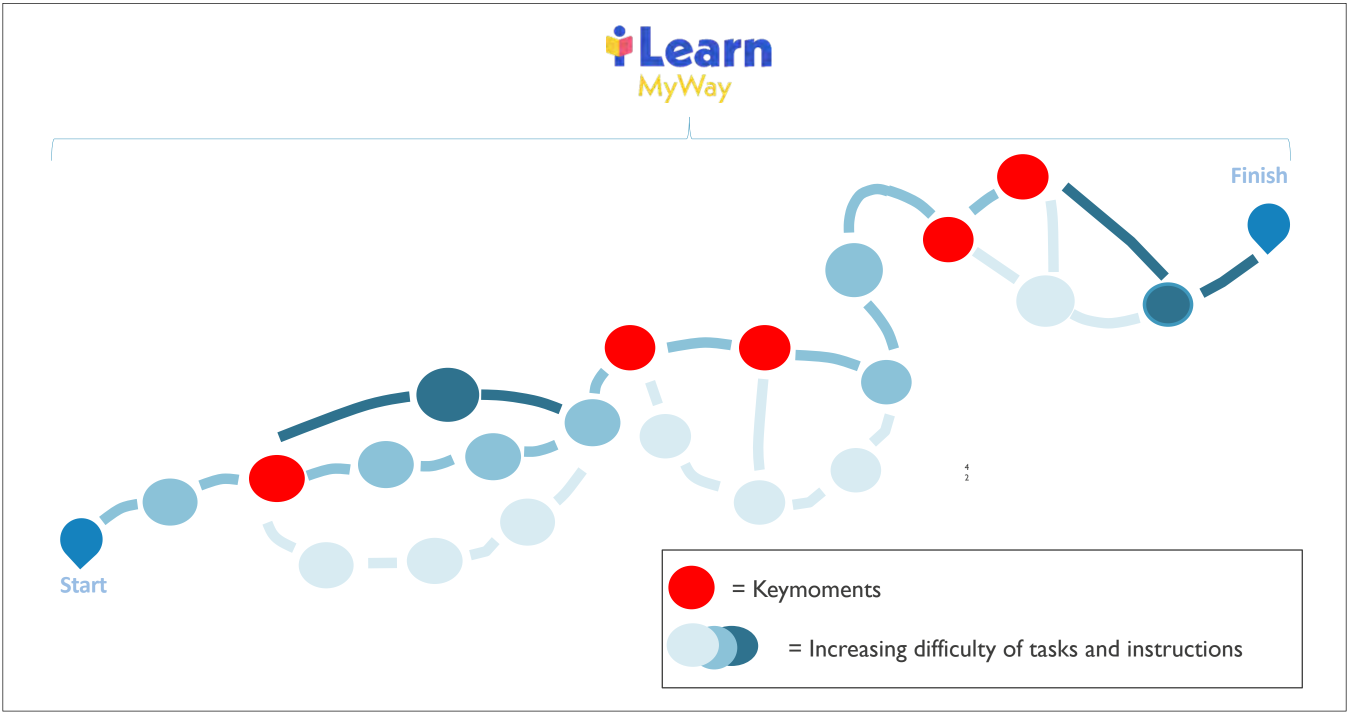
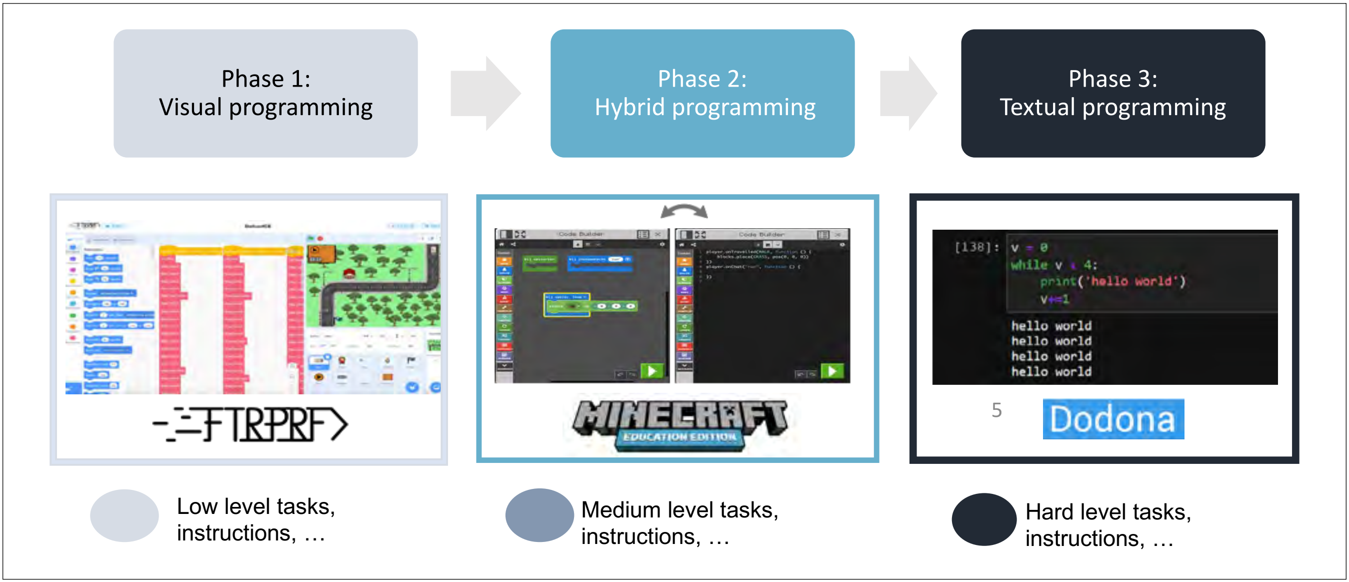


FIGURE 4. An overview of the DPL-track structure.



```
[138]: v = 0
       while v < 4:
           print('hello world')
           v+=1
       hello world
       hello world
       hello world
       hello world
```

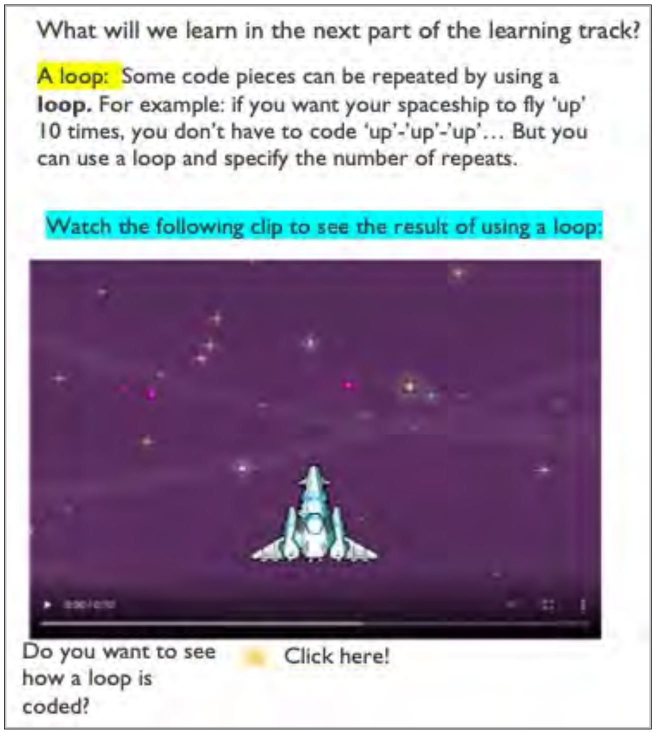
FIGURE 5. An overview of the learning environments: FTRPRF, Minecraft and Dodona.

development, the design challenges from the introduction (transition from visual to textual programming, personalization, and support teachers' role) will be considered. The goal of the DPL-track is to allow students to learn programming more easily and prepare them for learning real-life programming syntax. It also aims to support teachers in the delivery of programming education with adaptive learning materials.

EXPERIENCE OF THE DESIGN

GENERAL STRUCTURE

The DPL-track is a multi-pathway sequence of learning activities that consist of programming-related practice and instructions (see Figure 4). To offer personalized tracks based on student's individual needs, the DPL-track was built in i-Learn. I-Learn is a platform that aims for personalized learning in primary and secondary education. To realize this aim, i-Learn was established with a range of existing and qualified



What will we learn in the next part of the learning track?

A loop: Some code pieces can be repeated by using a **loop.** For example: if you want your spaceship to fly 'up' 10 times, you don't have to code 'up'-'up'-'up'… But you can use a loop and specify the number of repeats.

Watch the following clip to see the result of using a loop:

Do you want to see how a loop is coded?

Click here!

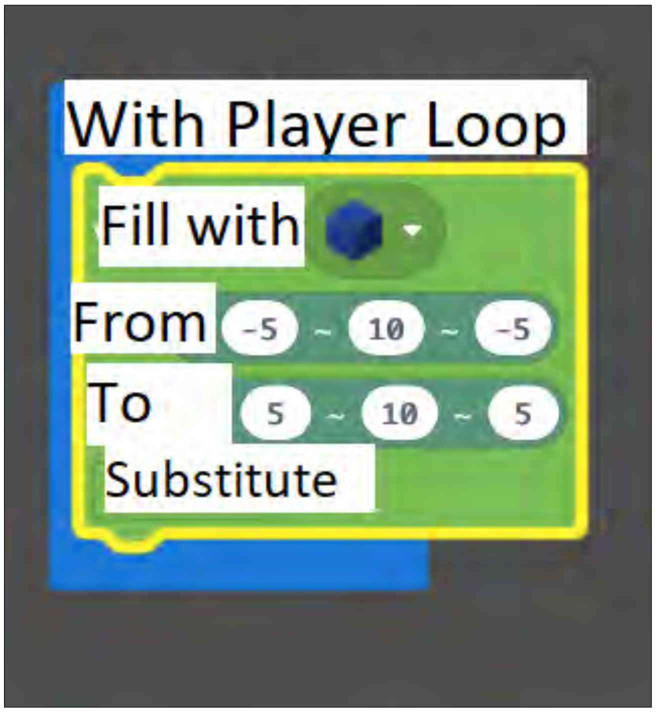
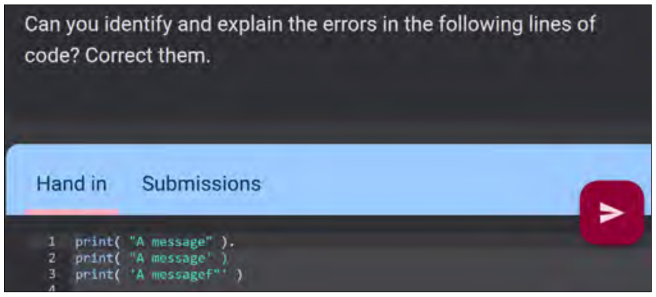


FIGURE 6. Example of an instruction in FTRPRF, focus on loops.

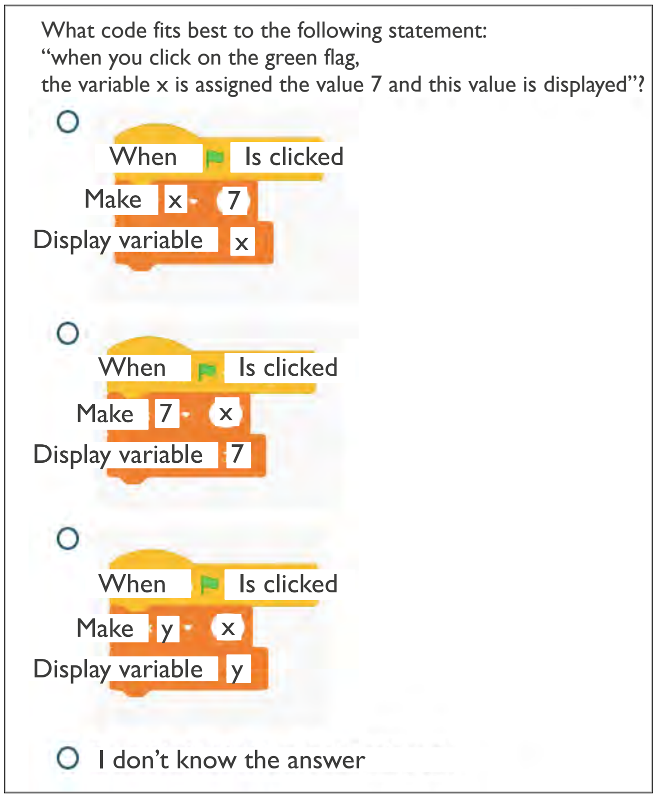
educational tools, activities within these tools, and learning tracks holding sequences of such activities. The i-Learn platform functions not only as a tool, activity, and learning track library, but teachers can also create or customize learning tracks themselves. All learning tracks offer built-in adaptivity facilitated through 'key moments' which mostly comprise quizzes or questions and consider cognitive, metacognitive,



Can you identify and explain the errors in the following lines of code? Correct them.

```
print( "A message" ).
print( "A message' )
print( 'A messagef"' )
```

FIGURE 7. Example of an exercise in Minecraft Education, focus on loops.



What code fits best to the following statement:
"when you click on the green flag,
the variable x is assigned the value 7 and this value is displayed"?

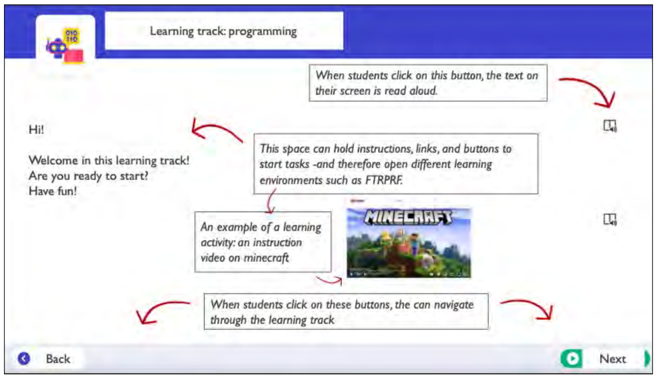
○
When Is clicked
Make x 7
Display variable x

○
When Is clicked
Make 7 x
Display variable 7

○
When Is clicked
Make y x
Display variable y

○ I don't know the answer

FIGURE 8. Example of an instruction in Dodona, focus on debugging.



Learning track: programming

Hi!

Welcome in this learning track!
Are you ready to start?
Have fun!

When students click on this button, the text on their screen is read aloud.

This space can hold instructions, links, and buttons to start tasks -and therefore open different learning environments such as FTRPRF.

An example of a learning activity: an instruction video on minecraft

When students click on these buttons, the can navigate through the learning track

FIGURE 9. Example of quiz question in key moment A.

or affective learner characteristics as a source to personalize the learning track.

Figure 4 illustrates a simplified overview of our DPL-track. We also use this simplified overview to explain design choices later in this paper. The Figure contains (a) key moments (red dots) facilitating adaptivity and (b) learning activities such as introduction videos, theoretical sections, programming tasks, and quizzes. The key moments guide students to different phases of varying difficulty (see the range of blue dots, also corresponding to Figure 5). These learning activities are further discussed later is this paper.

LEARNING ACTIVITIES

Learning activities contain either practice or instruction. Students practice through programming tasks. These programming tasks become unobtrusively more difficult, ranging from visual, hybrid, and textual programming. To create these tasks, different programming environments were selected (see Figure 5): (a) FTRPRF for visual programming, (b) Minecraft Education for hybrid programming, and (c) Dodona for textual programming.

Next to practice, there are instructions providing students with introductions, theory, tutorials, simulations, etc., through text or videos (see examples depicted in Figures 6, 7, and 8). Instruction does not only include information about the three learning environments, but also covers programming-specific knowledge with explanations of (a) basic concepts in CT used for programming such as decomposition, algorithmic thinking, and pattern recognition, (b) basic programming functions such as loops and iterations, and (c) basic Python syntax such as the functions *print()* and *len()*.

KEY MOMENTS

Key moments were integrated both at the start and throughout the DPL-track. When arriving at a key moment, students encounter (quiz) questions examining cognitive aspects (such as programming knowledge) or metacognitive aspects (such as self-estimation of programming knowledge). Based on their answers, students are unobtrusively guided to DPL-tracks with personalized instructions and tasks (see example quiz question in Figure 9).

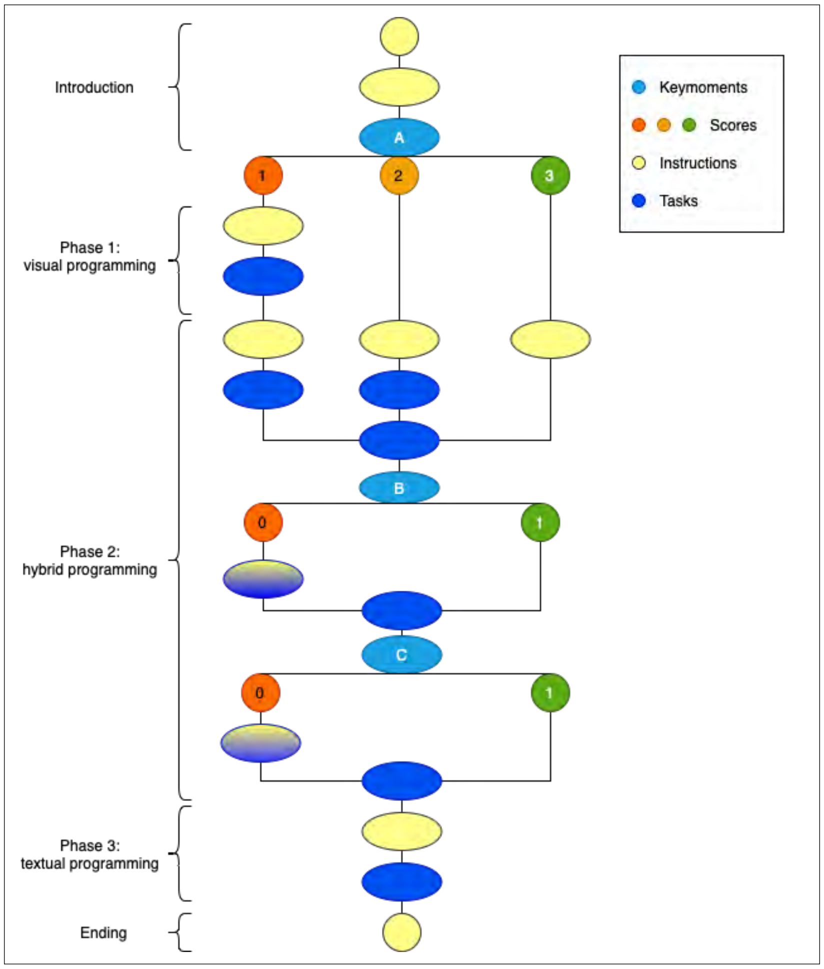
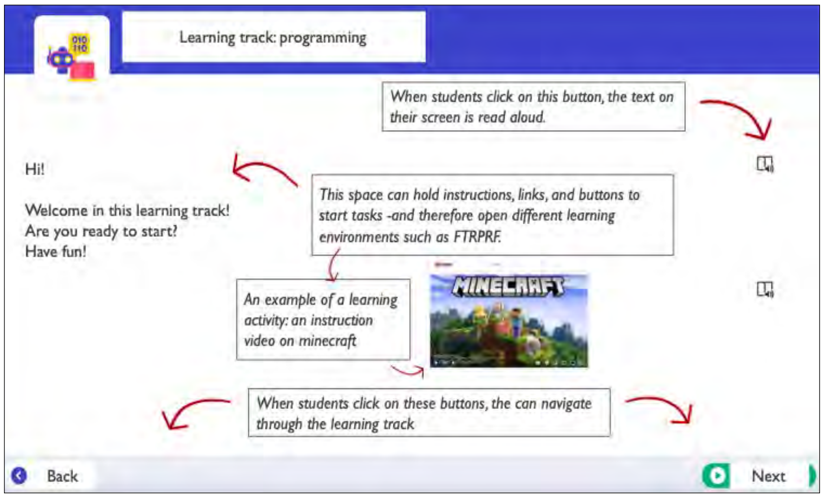

Introduction
Phase 1: visual programming
Phase 2: hybrid programming
Phase 3: textual programming
Ending
A
1
2
3
B
0
1
C
0
1
Keymoments
Scores
Instructions
Tasks

FIGURE 10. A more detailed version of the DPL-track.



Hi!

Welcome in this learning track!
Are you ready to start?
Have fun!

When students click on this button, the text on their screen is read aloud.

This space can hold instructions, links, and buttons to start tasks -and therefore open different learning environments such as FTRPRF.

An example of a learning activity: an instruction video on minecraft

When students click on these buttons, the can navigate through the learning track

FIGURE 11. The first slide of the DPL-track.

The sequencing of the DPL-track

Figure 10 provides a more detailed overview of the DPL-track. The different elements including learning activities (yellow and dark blue dots) and key moments (red dots in

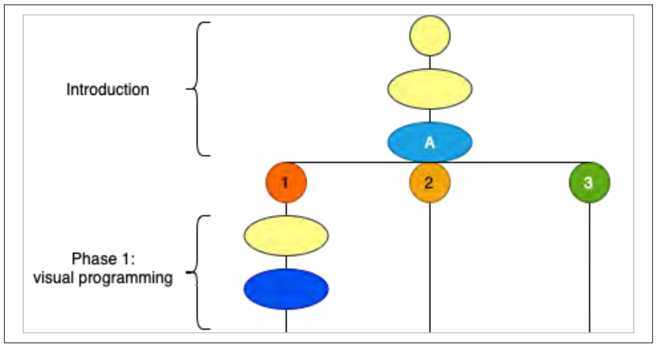


FIGURE 12. An overview of phase 1 of the DPL-track.

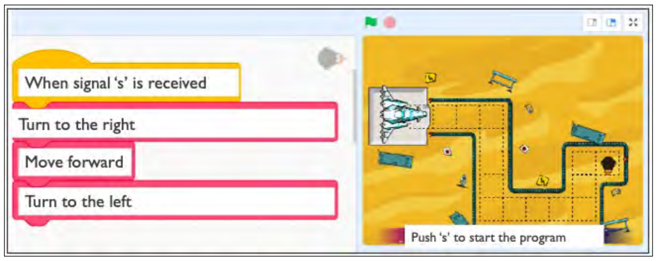


FIGURE 13. Example of an unfinished programming task in FTRPRF.

Figure 4, here blue) are explained later in this paper. Using this figure, the user experience is further described.

Introduction

When opening the DPL-track in i-Learn, students arrive at a general introduction. The learning material is displayed like a slideshow (see Figure 11) and presented via text, images, and videos. Students have the option to use a read-aloud function when encountering text. Other features are navigation buttons, by which students can operate through the slides. The slides occasionally contain practice buttons that lead to the learning environments FTRPRF, Minecraft Education Edition, or Dodona.

After a short introduction to CT and programming, students encounter 'key moment A' in which their prior knowledge regarding programming is tested via a short quiz. The quiz tests knowledge concerning sequence, conditions, loops, and specific Python syntax. For visual programming questions, a section of the Test of Basic Programming Concepts (Tsai, 2019) was modified. For the textual programming question, a section of the Prepared for Future Learning (PFL) test (Grover & Pea, 2013) was selected and adapted. After completion, instead of receiving scores, students are matched with a specific robot corresponding to students' cognitive level: Students who answered all questions correctly scored four points, were categorized as high-skilled, and were matched with robot So12 (see Figure 12 and 14). These students were sent to the introduction of phase 2 but then automatically skipped some low-level beginning

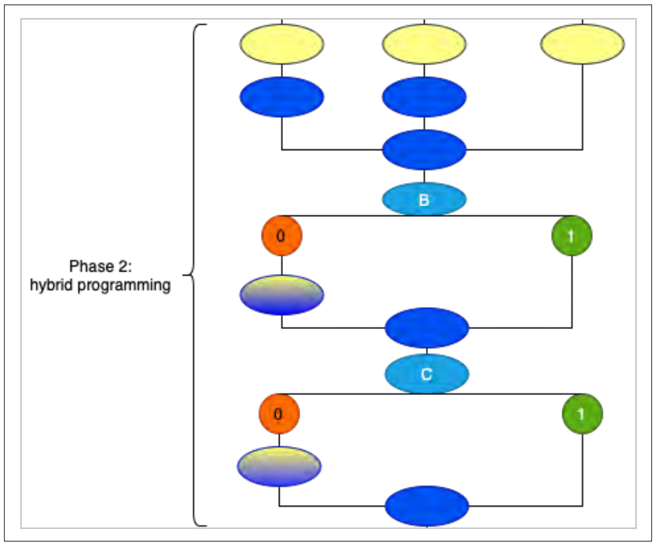
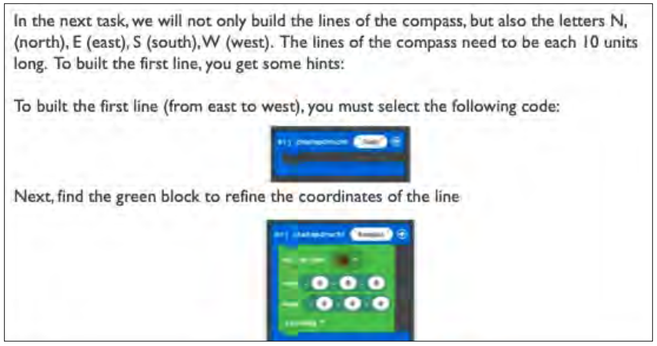


FIGURE 14. An overview of phase 2 of the DPL-track.



In the next task, we will not only build the lines of the compass, but also the letters N, (north), E (east), S (south), W (west). The lines of the compass need to be each 10 units long. To built the first line, you get some hints:

To built the first line (from east to west), you must select the following code:

Next, find the green block to refine the coordinates of the line

FIGURE 15. An example instruction in BookWidgets (visual programming).

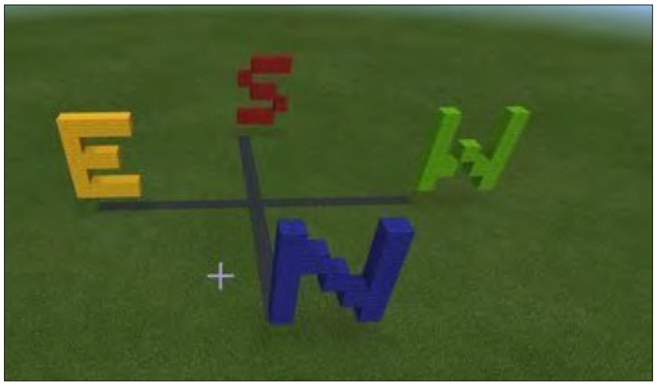
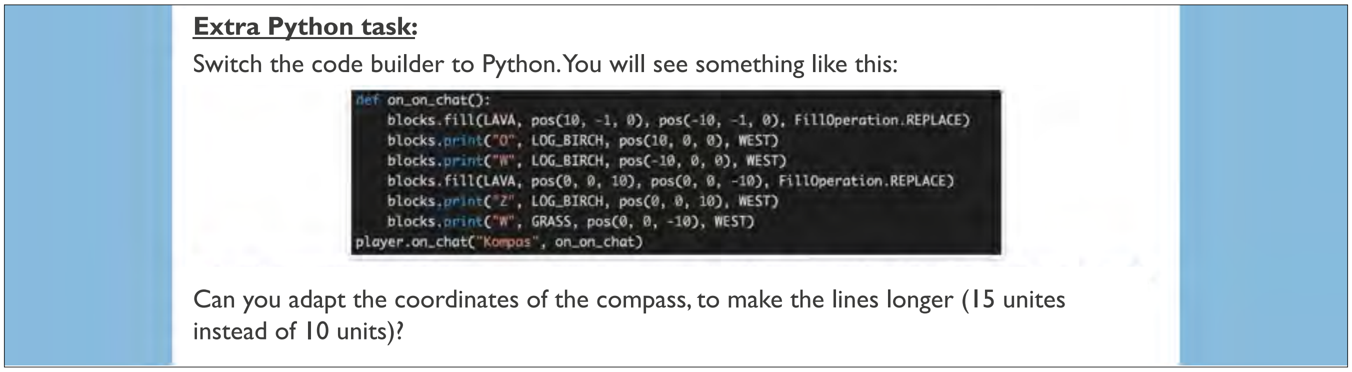


FIGURE 16. An example solution of a task in Minecraft.

tasks (see track 2 from Figure 14). Students who scored three points were considered medium-skilled. They were matched with robot YuZu and were sent to the beginning of phase 2. Finally, students with lower scores were matched with robot Ra28 and started at the beginning of phase 1 (see track 1 from Figure 12). Students are not aware of the level to which they are attributed.

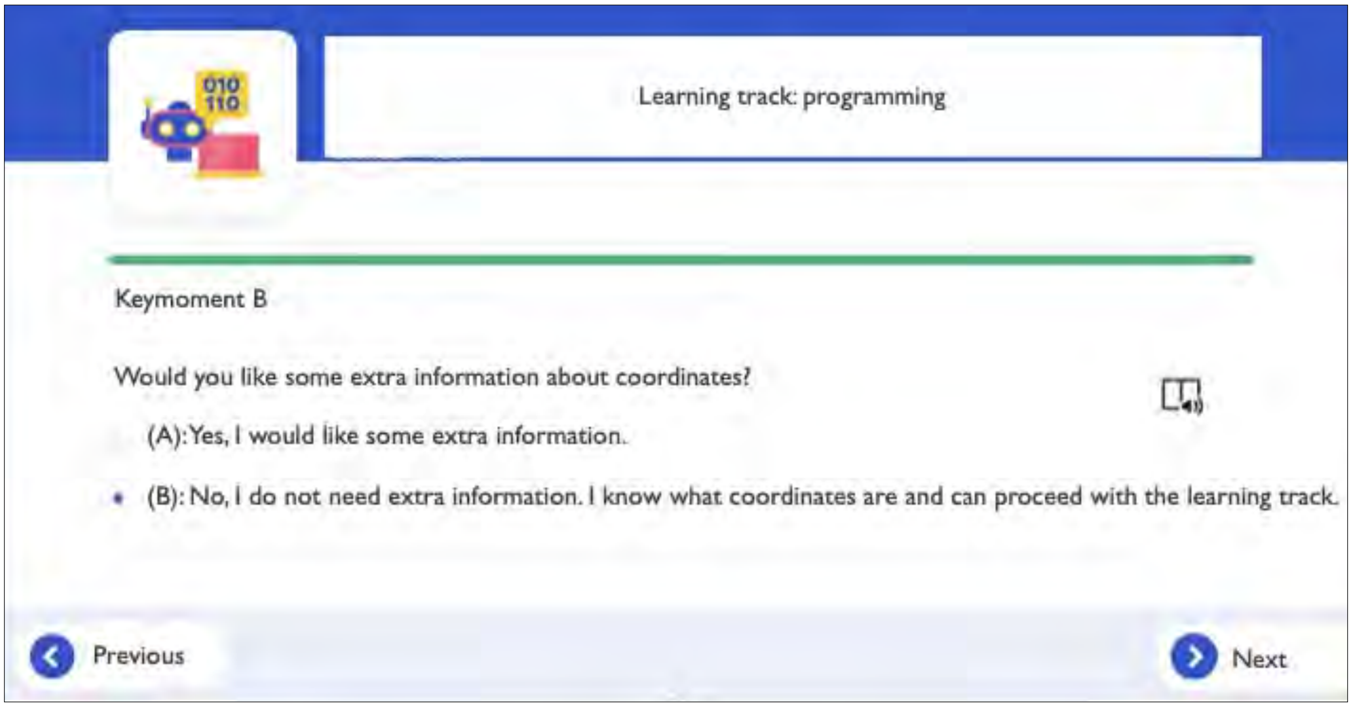

Extra Python task:

Switch the code builder to Python. You will see something like this:

```
def on_on_chat():
    blocks.fill(LAVA, pos(10, -1, 0), pos(-10, -1, 0), FillOperation.REPLACE)
    blocks.print("O", LOG_BIRCH, pos(10, 0, 0), WEST)
    blocks.print("W", LOG_BIRCH, pos(-10, 0, 0), WEST)
    blocks.fill(LAVA, pos(0, 0, 10), pos(0, 0, -10), FillOperation.REPLACE)
    blocks.print("Z", LOG_BIRCH, pos(0, 0, 10), WEST)
    blocks.print("W", GRASS, pos(0, 0, -10), WEST)
player.on_chat("Kompas", on_on_chat)
```

Can you adapt the coordinates of the compass, to make the lines longer (15 unites instead of 10 units)?

FIGURE 17. An example instruction in BookWidgets (textual programming).


Learning track: programming

Keymoment B

Would you like some extra information about coordinates?

(A): Yes, I would like some extra information.

- (B): No, I do not need extra information. I know what coordinates are and can proceed with the learning track.

FIGURE 18. Question during key moment B.

Phase 1: Visual programming

In phase one, students interact with the learning environment FTRPRF (see Figure 13). FTRPRF resembles Scratch and allows students to manipulate visual elements (blocks representing codes) when developing animations and games.

In FTRPRF, students are introduced to a built-in story as they learn about a group of avatars who want to go on a space mission. Practice and instructions relate to that engaging story: For example, when students learn about loops, they use loops to help guide an avatar to a spaceship. Phase 1 holds four lessons (or 'space missions') covering concepts such as algorithmic thinking, pattern recognition, abstraction, and includes tasks related to building general code structures and easy loops.

Phase 2: Hybrid programming

In the second phase (see Figure 14), students are provided with a summary of theory (programming concepts and meanings) from Phase 1. In doing so, students who scored medium or high on the prior knowledge quiz from key moment A do not miss instruction. After that recap, students move on to the second programming environment, i.e., Minecraft: Education Edition, which is an educational adaptation of the popular game Minecraft (Kuhn, 2018).

In Minecraft: Education Edition, students can explore procedurally generated lands and build block-based structures in a first-person game environment. Students can also press a button on the keyboard and switch from visual programming language to textual programming language (Python). For example, one task can be to build a compass via visual programming and then change the material or coordinates of the compass via textual programming (see Figures 15, 16, and 17).

Since Minecraft can only be used as a practice platform and does not allow for the presentation of instruction via visuals, we used an additional tool to present students with accompanying instructions. BookWidgets is a Belgian educational content authoring tool that can be used to present text, images, and videos (BookWidgets, n.d.).

During Phase 2 students are presented with key moments B and C that contain questions to self-estimate their programming knowledge (key moment B relates to coordinates and key moment C to variables, see Figure 18). According to their desire, students are presented with easier/harder tasks and more/less instruction.

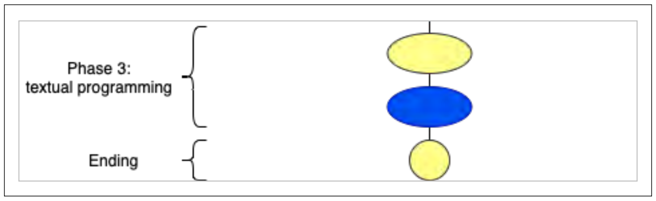


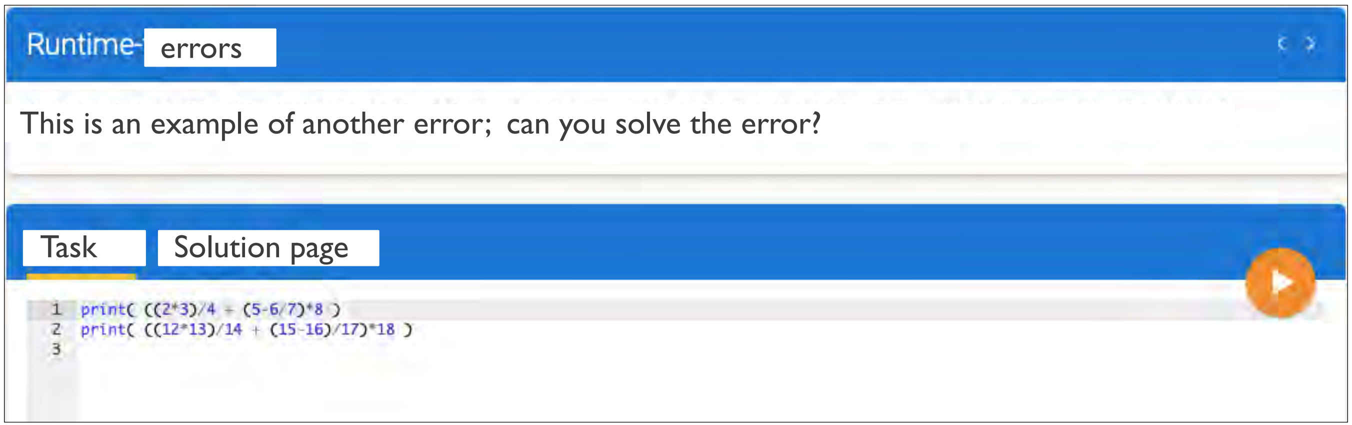
FIGURE 19. An overview of phase 3 of the DPL-track.

Phase 2 holds five lessons (or 'challenges') covering a summary of the learned concepts from phase 1 (algorithmic thinking, pattern recognition, and abstraction) and tasks related to events, coordinates, variables, iterations, and conditionals.

Phase 3: Textual programming

In the third phase (see Figure 19), students are provided with learning materials in Dodona.

Dodona is a textual programming learning environment (see Figure 20) resembling a more traditional developer environment (Van Petegem et al., 2022). It contains DPL-features such as automatic programming assessment (label pass/fail), advanced logging of student program submissions and time on task, and the presentation of lesson materials in programming with appropriate syntax highlighting (displaying components of code in colors according to categories of terms).



This is an example of another error; can you solve the error?

```
print( ((2*3)/4 + (5-6/7)*8 )
print( ((12*13)/14 + (15-16)/17)*18 )
```

FIGURE 20. An example task in Dodona.

DIFFICULTY	DESIGN CHALLENGE	DESIGN CHOICE	FRAMEWORK
Challenging learning material (Govender, 2006; Jenkins, 2004).	Transitioning from visual to textual programming	Scaffold students from using code to creating code and connect previous materials in visual programming to textual programming	Use-modify-create (Lee et al., 2011) and PRADA (Dong et al., 2019)
Large variation in previous knowledge (Salac et al., 2021).	Personalise learning materials according to previous knowledge	Add system-controlled cognitive adaptivity and learner-controlled metacognitive adaptivity	Vandewaetere and Clarebout, 2014
Difficulty in teaching (Brown et al., 2014)	Create materials to support teachers' role	Add system-controlled cognitive adaptivity and learner-controlled metacognitive adaptivity	Vandewaetere and Clarebout, 2014

TABLE 1. Overview design decisions and frameworks used.

Phase 3 holds four lessons (or 'challenges') covering all concepts from Phase 1 together with Python concepts such as 'syntax' and tasks related to basic functions such as print() to display text, len() to measure the length of lists as well as tasks related to debugging and variables.

The time to complete all three phases is estimated at four to five hours, depending on the student's prior knowledge and interests.

DESIGN PROCESS AND DESIGN CHOICES

The Design Process

Designing learning activities was a collaboration with software developers (from FTRPRF, Minecraft Education Edition, and Dodona). They provided existing learning materials and ideas, which were then substantially modified. For example, the selected FTRPRF learning material was intended for slightly younger students (around 7-10 years old) as it originally held 12 lessons and more low-level programming exercises. These lessons were shortened to four lessons while learning activities were rewritten to be more suitable for our target audience.

Design choices (see Table 1) are discussed in the following section. After developing the prototype, teachers and students evaluated it (see further in the failure analysis section).

Choices Based on Research Literature

Programming

Given the focus on the design challenges regarding the difficult shift from visual to textual programming, we based our scaffolding on previous efforts to scaffold from simple to complex. For example, we used the 'use-modify-create' framework, which represents a pattern of engagement and is often used as a basis for support in multiple programming courses (Lee et al., 2011). This three-phased framework comprises a beginning phase in which students 'use' pre-built blocks of code or applications. Over time, students move to the second phase to 'modify' the visual programming creations with textual programming, usually by first changing simple parameters. In doing so, they gain an understanding of programming and develop new programming skills. Finally, in the last phase, students 'create' new materials, functions, or even artifacts. The three phases are based on the premise that these aids for sequentially acquiring knowledge, termed scaffolding, strengthen the acquisition of programming and work to reduce anxiety for novice programmers (Lee et al., 2011). We applied the 'use-modify-create' framework in the DPL-track to help students retain and apply new knowledge while moving from visual to textual programming language. When students enter the second phase and open Minecraft Education Edition, they first encounter visual programming tasks, in which they can use pre-built blocks to construct a program ('use'). After many practice opportunities, students move on and encounter hybrid programming tasks. These allow students to practice and modify existing visual codes, by altering parameters in the textual interface and demonstrating the similarities between blocks and textual programming ('modify'). After Minecraft, students navigate to Dodona where they build programs ('create'). Following this structure, the DPL-track provides scaffolding as described in the 'use-modify-create' framework.

Next to the 'use-modify-create' framework, we also selected the Pattern Recognition, Abstraction, Decomposition, Algorithms (PRADA) framework (Dong et al., 2019) which aims to describe and clarify specific subskills of CT to increase integration in K-12 classes. In the design, the PRADA-framework was consulted to connect disparate programming concepts and explain their use for students no matter the language. The subskills in the PRADA-framework are:

- Decomposition: Analysing a problem to break it down into smaller parts.
- Pattern recognition: Recognizing patterns from data.
- Abstraction: Identifying the underlying principles that generate observed patterns.
- Algorithms: Creating clear, step-by-step instructions for solving problems.

These subskills were used to structure lessons in each of the three phases by connecting previously understood visual programming material (for example the coding of loops or repetition of code—related to the subskill pattern recognition- in FTRPRF) to a new textual programming language (loops in Dodona). As such, connections were made to explain syntax and useful commands in textual programming languages with direct comparisons to previously learned visual programming languages.

By consulting both frameworks during the design of tasks and structure, the design principle 'a gentle transition from visual to textual programming' was pursued.

DPL

Given the focus on the two design challenges regarding 'personalization to meet differences in students' knowledge and interests' and 'supporting the teacher role', we again consulted literature to build on. To think about different types of adaptivity for the key moments, we used the framework of Vandewaetere and Clarebout (2014)—which describes four dimensions of adaptivity. As we describe the outcome of the design according to the framework, a summary is given: In sum, adaptivity can differ according to target, method, time, and source (Vandewaetere & Clarebout, 2014). With respect to (a) target, the framework identifies targets of adaptivity

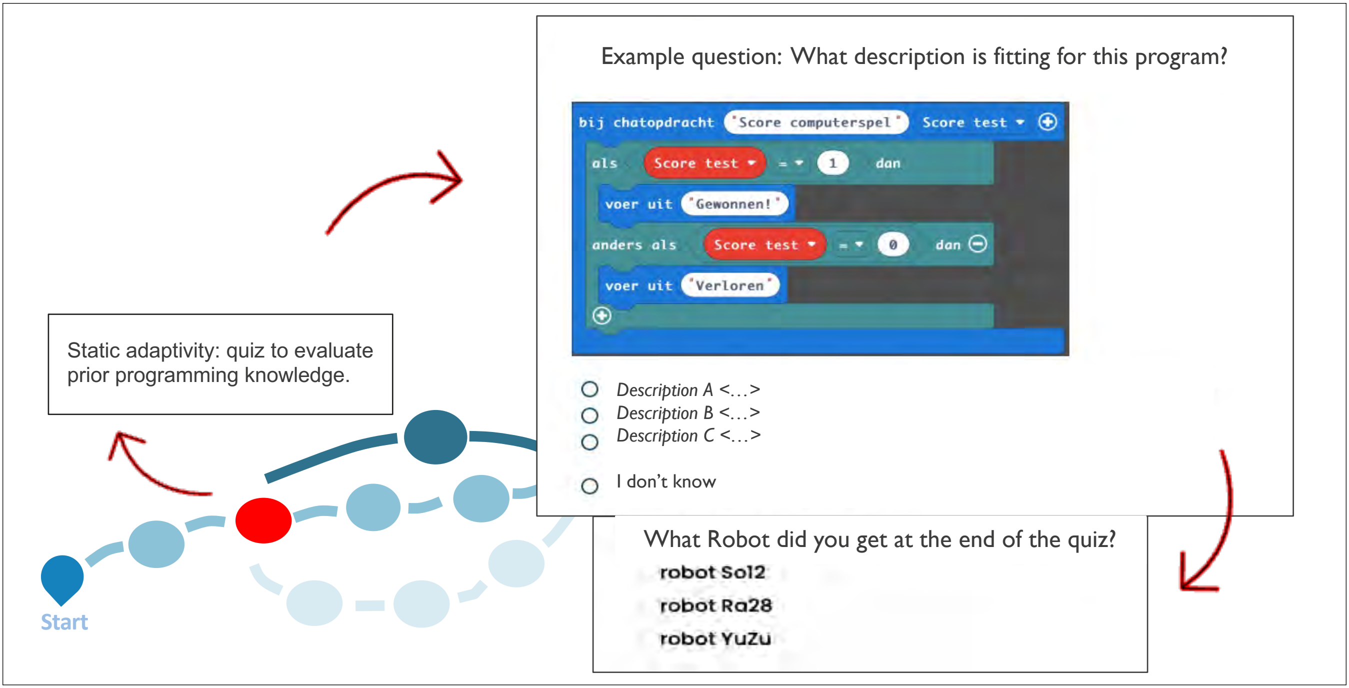

Example question: What description is fitting for this program?
bij chatopdracht "Score computerspel" Score test
als Score test = 1 dan
voer uit "Gewonnen!"
anders als Score test = 0 dan
voer uit "Verloren"
Description A <...>
Description B <...>
Description C <...>
I don't know
What Robot did you get at the end of the quiz?
robot So12
robot Ra28
robot YuZu
Static adaptivity: quiz to evaluate prior programming knowledge.
Start

FIGURE 21. Key moment A within the DPL-track.

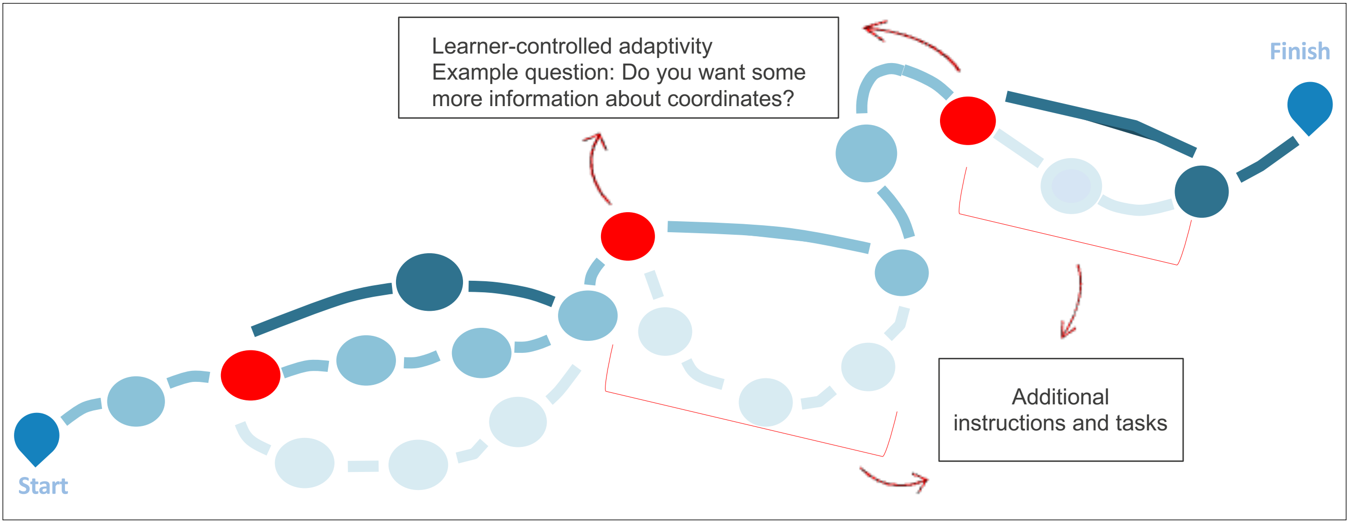

Learner-controlled adaptivity
Example question: Do you want some more information about coordinates?
Finish
Additional instructions and tasks
Start

FIGURE 22. Key moment B and C within the DPL-track.

such as level of tasks, presentation format, or degree of instruction. Regarding (b) method, different methods of how adaptation is initiated are identified e.g., by the learner (learner-controlled) or system (system-controlled), or both. For (c) time, the framework refers to when the adaptivity takes place, which can be before (static), during (dynamic) the learning activity or both (dual pathway). Finally, (d) the source relates to different origins of adaptivity such as cognitive or metacognitive learning characteristics, but also learner-system parameters such as time spent on a task.

To acknowledge students' prior knowledge and interests (beginners as well as experienced programmers), three key moments (A, B, and C) are integrated which guide students to personalized tracks. Key moment A (see Figure 21) focuses solely on prior knowledge assessment (cognitive learner parameter as the source of adaptivity) at the beginning of the track (static adaptivity) and leads students to separate tracks within the DPL-track in which they are provided with different learning environments, support/instruction and difficulty of tasks (adaptation of content, presentation and support/instruction). Students cannot change this track (program-controlled adaptivity).

As we aimed for a variation of adaptations (not only system-controlled and focussed on cognitive learner parameters), key moments B and C (see Figure 22) were added which hold metacognitive questions (source of adaptivity: metacognitive learner parameters). If desired (learner-controlled), students get both additional instructions and tasks (adapted content and support/instruction) before moving on to harder challenges. Therefore, both key moments add to learner control for adaptivity in the DPL-track (Vandewaetere & Clarebout, 2014).

By providing key moments both before (A) and during the DPL-track (B and C) dual pathways and shared-control adaptivity are pursued (see Vandewaetere & Clarebout, 2014). Besides the key moments, there are other features that contribute to adaptivity. For example, the i-learn platform contains a button that can be activated by students to turn written text into audio (student-controlled adaptivity). Another example is a feature within Dodona, which facilitates automatic programming assessment (program-controlled). Personalization of tasks or support/instruction does not only meet differences between students but can also support teachers when teaching heterogeneous classes (Holmes et al., 2018; Major & Francis, 2020). Tools can assist teachers by immediately tailoring the learning process. Furthermore, when collaboration or synergy occurs between teachers and tools, students receive multiple forms of scaffolds which can enhance the learning process (Tabak, 2004).

By considering manifold adaptivity during the design of tasks and structure, the design principles 'personalize learning experiences for students' and 'support teachers in their teaching practice' were pursued.

FAILURE ANALYSIS

Evaluation of the DPL-track

As the development of the DPL-track reflected DBR principles (in which "development-evaluation-refinement" is the key development process), we focused heavily on evaluation. In doing so, the goal was to improve the design both motivational and functional by assessing seven aspects:

- General: How did they like the DPL-track?
- Learning content: What did they think about the provided theory?
- Tasks: What did they think about the provided tasks?
- Clarity of learning goals: Did they think the main purpose of the DPL-track is clear enough?
- Support/instructions: What did they think of the provided support and instruction?
- Evaluation: How did they experience the quiz questions during the key moments?
- Personalization: How did they like the built-in-adaptivity?

We tested the track in two phases: first with students, and then with teachers. We mainly wanted to see if the DPL-track matched teacher and student expectations, which was a concern given the variation in previous programming experience. With 17 students, we observed their use of the DPL-track, asked ad-hoc questions during their use, and then conducted two focus groups guided by questions on the previous seven evaluation aspects (See Appendix A for interview protocol). One focus group had four high-skilled students, and another had thirteen low to medium-skilled students. Students were enrolled in the first grade of secondary education and were majority male (only three students were female). These students were recruited from an after-school coding camp and had diverse levels of programming knowledge, skills, and interests.

In the second cycle, we individually interviewed 9 ICT/STEM teachers after they evaluated the DPL-track on their own. Interview questions again followed the major themes previously mentioned. For interview protocol, see Appendix A. All teachers received a dummy i-Learn student account to test the DPL-track and a digital manual encompassing all practice and instructions from the track. The selection of teachers was also diverse: One highly experienced teacher frequently taught programming and even developed a syllabus himself. He was categorized as high-skilled. Six teachers had some programming experience and were accordingly categorized as medium-skilled. Finally, two teachers with no programming experience were categorized as low-skilled.

Evaluation by students

Hereafter, we describe how students evaluated the design along the previous themes.

General Evaluation

We observed that students experienced the DPL-track as engaging, supportive, and challenging. They confirmed these positive perceptions multiple times during the discussions.

Learning Content

Low-skilled students (who start in FTRPRF) liked the engaging story of the space mission which guided them through the learning content. Students applauded FTRPRF for the clear explanations and summarizations of basic theoretical concepts. However, as a point of improvement, students explained that a video or more visuals should reinforce textual explanations of theoretical concepts, specifically abstract concepts (such as algorithmic thinking). The medium- and high-skilled students (which started in Minecraft) indicated explanation and repetition of concepts could be better and more frequent (e.g., proper explanation of 'textual and visual programming', 'syntax'). In Dodona, the high-skilled students sometimes struggled with learning new functions. Particularly in the last exercise, which required students to apply a variety of previously learned functions. We saw in log data that the students submitted 20 coded programs in total, 9 of which were syntactically correct, but failed to use all learned functions at the same time. With simple reminders of functions, such as the str() function, and group collaboration, each student eventually succeeded in the exercise. As such, the extra focus should go to introducing -and especially repeating- new functions (e.g., 'Len', 'input') and function combinations since some explanations were sometimes too abstract.

Tasks

Task-related feedback was positive. Low-skilled students perceived FTRPRF as a supportive learning tool that they enjoyed practicing. All students were positive about the textual programming challenges in Minecraft (switching between block to text-based programming) as they acknowledged writing textual code on their own would be too difficult. Concerning Dodona, high-skilled students found the tasks very challenging. Sometimes even too hard. During those tasks, we observed the occurrence of spontaneous group work, especially for debugging: students helped each other to solve bugs and read code. This is positive since it is reminiscent of real-world programming projects (e.g., software engineering, scientific code analysis, etc.).

Clarity of Learning Goals

When discussing clarity of learning goals, students were often unaware of what they were learning and why the tools were selected for the acquisition of programming. We had to elaborate on this topic and challenge them to reflect on it. Once we clarified the learning goals, students could identify why we selected the tools. As one student clarified during the discussion: "I like the fact that we go from a 2D world in FTRPRF to a 3D world in Minecraft, it is harder to code but as you grow, this challenge is very welcome." Another student positively elaborated on the gap between visual and textual programming: "I cannot write textual code on my own. I become easily frustrated. So, it is really nice that we don't have to do that, and we can always rely on the blocks to learn more about Python."

Instructions and Support

Students indicated instruction/support should be increased. For technical support, they would like more introductory videos on how to engage with i-Learn and the learning environments (e.g., setting up a Minecraft world). Students found the combination of instructional pictures and videos, as applied in the Dodona track, most useful. For instructional support, students liked videos about how to build code, how to debug, and how to solve tasks. However, students indicated that programming jargon, such as bytes and syntax, lacked proper explanation. For instructional support, it was noticed that students' freedom of choice' was not always encouraged enough. For example, one student asked if he could already use loops in FTRPRF. Another student asked if he could pick another color to build a compass in the Minecraft world. Nonetheless, we also observed students taking advantage of the 'freedom of choice' as they were, for example, wandering off or using their functions to alter the environment which made it hard to continue with the other tasks. Next to general instructional support, we also reflected on task solutions provided throughout the DPL-track. Most students used the solutions responsibly. In Dodona, students liked the automatic feedback feature (which visualizes correct/incorrect and error types). Students repeatedly used this feature, with multiple submissions on every exercise to test newly written code. However, concrete solutions were missing. Students suggest this could be included as a last slide at the end of every mini-lesson.

Evaluation

For evaluation, all students completed key moment A. Log data from the low- and medium groups (n=13) showed that seven started in the first phase and six in the beginning of phase 2. For the high-skilled group, who were supposed to have prior knowledge about Python, two were categorized as high-skilled and two were not. Yet, together with the observations, we concluded the test made a good prediction, as they were less skilled programmers and often had to rely on the two stronger students to debug code and answer programming questions. During the discussion, the students told us they really liked the robots that were assigned to them after the quiz (not knowing which scores they resembled). They also liked the other key moments but did not pay much attention to them as they thought the key moments were just regular quiz questions. Log data indicated students engaged positively with the self-evaluating questions: we had no cases of students who overestimated themselves in 'key moments B or C' and ended up in improper tracks.
Evaluation by teachers

Teachers had two weeks to go through the complete DPL-track via a dummy i-Learn student account. During the test weeks, we remained available for support. All difficulties, questions, and suggestions as communicated by the teachers were registered. Finally, the teachers were invited for an individual interview (approximately 45 minutes). The same topics used for student evaluations were targeted, including the additional topic of adaptivity. In doing so, we wanted to increase understanding of how teachers perceived the DPL-track, both motivational and functional.

General Evaluation

Teachers were enthusiastic about the DPL-track and considered it useful. The the built-in adaptivity was especially positively valued.

Theory

Teachers (often medium-skilled) appraised the selection of theory. They applauded the sequence, flow, and generally growing difficulty of concepts. Apart from small content remarks -such as the replacement of difficult programming verbiage or simplification of definitions- most teachers found the theory clearly explained. However, not all teachers agreed on the theory selection: the two low-skilled teachers found some content too challenging (e.g., text-based programming concepts were found to be too difficult), while the high-skilled teacher was convinced that the difficulty level could be even extended. All teachers expressed the need for more visuals (e.g., pictures and videos) and repetition of new concepts.

Tasks

Teachers praised the alternation and variation from visual to textual programming. Some were already familiar with visual learning environments, such as Scratch (FTRPRF), but could not connect to more abstract text-based (or hybrid) program environments. They were happy to get to know new tools via the DPL-track. Many teachers provided us with some helpful refinement suggestions to optimize tasks: One teacher shared a self-made Minecraft task concerning 'variables' which was a better fit than the original task in the DPL-track. Another teacher suggested the addition of text-based Minecraft challenges at the very end of the DPL-track for students who finish early. One teacher also suggested adding more context/story to the Dodona track and helped with the development of an engaging story, which also fit the abstract characteristics of the tool. Most teachers found the tasks to be just the right amount, nicely developed, and complementary to learning subjects from other courses (e.g., coordinates within mathematics courses). However, not all teachers liked the distribution of block-based and text-based programming tasks. Again, two sets of opinions could be noticed: The high-skilled teacher would reduce the visual programming part, while the low-skilled and some medium-skilled teachers would expand it and reduce textual programming tasks.

Clarity of Learning Goals

Teachers considered the learning objectives to be transparent enough. As one teacher said: students will be aware of the main objective, i.e., learning to program, without having to focus on all the sub-learning objectives (such as learning about new theoretical concepts). Some teachers suggested adding a comprehensive summary (with sufficient visualizations) about the main learning objectives at the beginning and the end of the DPL-track, as they noticed from their practice that this helps to make the learning objectives extra clear.

Instructions and Support

Regarding instructions and support comments frequently mentioned: (a) Students should be provided with more technical support about how to open and use tools. (b) The presumption that all categorized low-skilled students know nothing about programming can demotivate the ones who know some basics. For example, if students know something about loops, they are not encouraged enough to use loops in the first tasks of Phase 1. (c) Teachers liked the instruction videos, as they were found to be clear and on a student level. They would like to see more of them throughout the DPL-track. Next to these three suggestions, teachers were also asked to give feedback on the amount of provided task solutions. Opinions were two-fold: some teachers found them to be useful and applauded the gradual decrease towards the end of the DPL-track. They value task solutions as students learn a lot from them, especially when building codes. Some teachers, however, were concerned students would take advantage of the provided solutions, i.e., copy-paste to solve tasks. For them, the solutions/examples are too easily within reach. This raised concerns about 'gaming the system' (cheating). One even suggested a point system to 'punish' students when they consult solutions.

Evaluation

Concerning evaluation, teachers shared similar thoughts: (a) key moment A is well constructed. They said the questions are relevant and nicely sequenced. Teachers liked the first question where students estimated their level, but also liked that this did not influence the score to avoid mismatch. One teacher referred to the risk of the Dunning-Kruger effect, which comprises an overestimation of skills due to a lack of knowledge. During the questions, low-skilled students are not discouraged as the difficulty gets acknowledged (i.e., "Do not worry if you cannot solve this puzzle. This is normal if you have no/little programming experience!"). Not showing the scores was also found to be positive. Instead, the robots were acknowledged as a fun way to reward and engage the students. (b) Teachers explained they missed the opportunity to learn from key moment A. They recommend going back to specific questions when the corresponding topic is covered, as an opportunity to learn and acknowledge progress. In addition, most also expressed the need for a final key moment as an end evaluation.

Adaptivity

Finally, adaptivity was considered valuable in the DPL-track as all participants reported on the same challenge: finding learning material fitting for heterogeneous student groups with different programming knowledge, competencies,

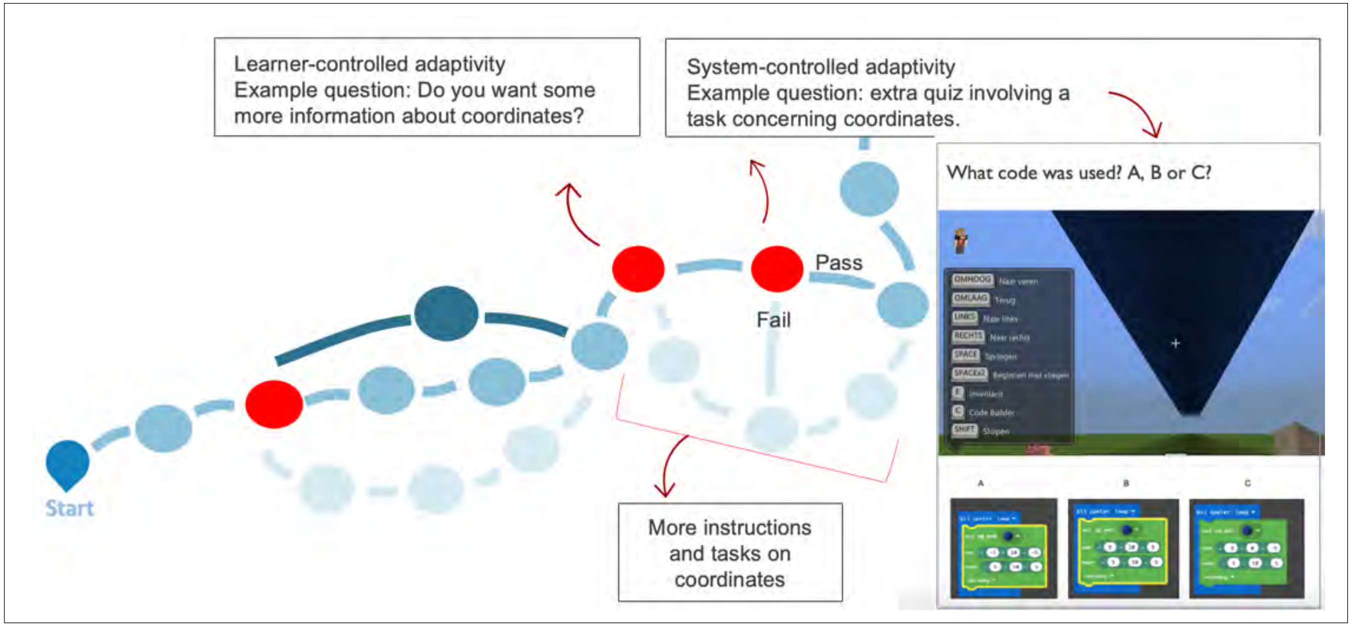

Learner-controlled adaptivity
Example question: Do you want some more information about coordinates?
System-controlled adaptivity
Example question: extra quiz involving a task concerning coordinates.
What code was used? A, B or C?
Pass
Fail
Start
More instructions and tasks on coordinates
A
B
C

FIGURE 23. Adjustment for next iteration—addition of an extra key moment after key moment B.

and motivations. They like that the DPL-track encourages and engages students as it not only helps the low-skilled students or challenges high-skilled students, but also that it provides some extra time for the teacher to further manage and guide the class. Teachers liked the student-controlled adaptivity (self-evaluation key moments B and C). However, most stated that system-controlled key moments (more specific questions to evaluate knowledge) should follow up the self-evaluation to correct for self-overestimation of students. Although the DPL-track only holds three key moments leading to adaptivity, many teachers explained this already exceeds expectations. They state that more moments would maybe become a burden (instead of supportive), as they would lose an overview of all tracks.

Reflection on Evaluation

To optimize the DPL-track, we compared the student discussions, observations, and log data to the teacher interviews, noting similarities for improvement. In doing so, six concrete plans for refinement were decided on and three design dilemmas were unraveled. These refinements do not contradict the major design principles (ease the transition from visual to textual programming, personalize learning experiences, and support the teachers' role), but concern the operationalization of these design principles. An overview of the plans of refinement related to teachers' and students' comments can be found in Appendix B.

Plans for refinements

The plans for refinement comprise five smaller functional design changes and one more fundamental change in structure. Starting with the smaller changes, the first improvement goal is to provide more balance between textual and visual support/instruction for practice and instruction. We will reinforce several parts of the DPL-track by adding supplementary videos and images. The second improvement goal includes more repetition and clarification of theoretical concepts, especially in Phase 2 and 3. In addition, programming-specific vocabulary without proper explanations will also be removed or adjusted. The third improvement goal relates to the clarification of learning goals. A brief overview of (a) explicit learning goals, (b) upcoming software tools and (c) their added value will be added at the beginning and end of the DPL-track. The fourth improvement goal will address the challenge of combining several tools (BookWidgets, FTRPRF, Minecraft, Dodona) in one track (in i-Learn). Students are expected to alternate between different screens/tabs. This can be a problem for students with, for example, low ICT skills. More technical support will be provided during transitions. The fifth improvement goal relates to key moment A: Currently, there is a missed learning opportunity, as students are not provided with answers or feedback when answering quiz questions. This was a deliberate design choice, as feedback (alongside corresponding scores) could hamper the unobtrusive guidance to a personalized DPL-track and therefore also demotivate beginning students with no programming knowledge. An effort will be made to insert some 'recall' learning moments (that also act as feedback) throughout the DPL-track.

The last improvement goal comprises a more fundamental change in the structure of the DPL-track, more specifically regarding key moments B and C. The teachers appreciated these students' self-evaluation moments (student-controlled adaptivity) but missed quiz questions for verification (system-controlled adaptivity). This verification should correct for students' self-overestimation (see also Dunning-Kruger effect) and assess whether students' programming skills are sufficient to proceed. We will add two additional key moments (one after key moment B and one after key moment C; see Figure 23) to check whether students who do not wish to receive extra information or tasks, really understand the learning content. If not, they will also go through extra instruction and tasks. Thus, these additional key moments will contain system-controlled adaptivity based on cognitive learner parameters. In doing so, we will create two shared-controlled adaptivity sets.

Three design dilemmas

In addition to the refinement plans, some design dilemmas are acknowledged. The first dilemma relates to differences between teachers' programming knowledge and their expectations: higher-skilled teachers can support students during more challenging tasks, while lower-skilled teachers cannot and therefore expect easier tasks. Thus, their programming knowledge influences the difficulty of students' tasks and instructions. Since most participating teachers were medium-skilled, we kept the level of difficulty for further iterations. However, we acknowledge that even when using DPL-tools -where the level of tasks is already adapted to students' knowledge-, there can be a dilemma of selecting tasks and instructions as different expectations can still occur.

The second dilemma concerns the provision of solutions at the end of every task. Some teachers valued it, while others did not (e.g., difficult to prevent students from gaming the system). This dilemma relates to instructional disobedience, described by Elen (2020) as a phenomenon that "occurs when learners do not act as expected from them in a learning environment." As reducing student control may not fully counter instructional disobedience (Elen, 2020) and the emphasis for this design was on learning through the act of programming, we decided to keep the solutions in the DPL-track as a learning tool instead of reducing/limiting them. We will, however, foresee more instructions related to the goal of the solutions, to optimize understanding (and acceptance) of their instructional expectations. Nevertheless, we understand that it is unrealistic to assume only instructions elicit exclusively compliant students, and therefore acknowledge providing solutions implies an extra responsibility for students (to not misuse them) and teachers (check for students' misuse).

The third dilemma - also related to instructional disobedience - pertains to the freedom of choice within programming tasks (e.g., coding with different blocks and colors) and learning environments, especially within Minecraft: education edition (e.g., wandering around in the never-ending world. On multiple occasions, students are encouraged to try things out. However, instructionally encouraging students' freedom of choice can invoke a risk of instructional disobedience. Again, this causes extra management challenges for teachers. We value freedom of choice during programming as it can boost students' motivation (Autio et al., 2011). To avoid unmotivated starters, more differentiation regarding instructions will be added (e.g., "If you know how to use loops, feel free to use them!" or "You can also choose a block and color of your own choice"). However, students will also be encouraged to focus and follow instructions by, for example, changing the mode of Minecraft from 'creative' to 'flat' to avoid distraction. In this 'flat' mode, students'-built objects do not interfere with NPC's or other pre-built objects.

CONCLUSION

General

Programming proliferates in education. However, it is not the easiest subject to teach and learn. Students often struggle to move from visual programming to textual programming and there are great differences between their programming knowledge and competencies. Likewise, teachers find it difficult to teach this relatively new subject to heterogeneous classes. As these challenges mirror the trends in the greater educational landscape, this design case focused on the design and development of a DPL-track with a twofold basis: programming and DPL. Overall, the three major design principles (ease the transition from visual to textual programming, personalize learning experiences, and support teachers' roles) were well received: Results indicate a variety of students experienced the DPL-track as engaging, supportive, and challenging. They acknowledged the value of hybrid programming and understood the benefits of using it to bridge the gap between programming languages. Teachers were also generally enthusiastic about the DPL-track. Of note, the built-in adaptivity received many positive comments, as it is found to be generally difficult to match different programming tasks to different students within a diverse class group. Further, we present findings and reflections for practical and theoretical stakeholders.

SIGNIFICANCE

The DPL-tool

The development of the DPL-track was centered around the use of existing tools and balanced theoretical choices with practical considerations from teachers and students. Using existing tools was a conscious decision. Software developers' perspectives and experience helped inform the design, which likely improved ease of use and usefulness for teachers and students. Nevertheless, these perspectives were also grounded in theory: while we used materials and tools offered by the software developers to teach programming, we also substantially changed the structure/organization to adhere to the 'use-modify-create' strategy and the PRADA-framework (Lee et al., 2011; Dong et al., 2019). These adjustments were supplemented with DPL-adaptations from the framework of Vandewaetere and Clarebout (2014). We considered these theoretical perspectives as complementary, as CT perspectives provided the thought processes needed in using programming in new challenges, programming education perspectives provided structures with which to learn programming, and DPL-perspectives provided the ability to proactively support students at every level. This combination resulted in a holistic, personalized design for teachers and students to test. Often, teachers receive few opportunities to be co-designers of educational technology. They use existing, pre-developed tools to teach, and adapt their pedagogy and the way the technology is used to suit their ends (Bunting et al., 2021). However, teacher-developer collaborations could benefit the development of innovations and encourage teachers to implement the innovation in their classrooms (Groff, 2017). In light of these considerations, we -as educational designers- will focus even more on collaboration with teachers in our future design efforts.

Insights as developers of educational technology

Based on the development process, we espouse the utility of evaluations with teachers and students. In this design case, it was very valuable to see the track in action during observation moments. The evaluative observations and discussions became valuable 'co-learning' moments. The discussion and interviews revealed differing expectations and needs, a common consequence of human-centered design approaches (Dimitriadis et al., 2021). Calls to include teachers and students through co-development in educational technology are ever present (Dimitriadis et al., 2021). In addition, we found the use of observations very interesting, as they are very valuable but often overlooked in the creation of educational technology. In this design case, formal and informal observations gave insights into the needs of users to optimize the user experience, providing evidence for assessments and challenges that might be missed by a teacher. When involving teachers into the DPL-track development, it showed that they applauded the adaptivity as it was supportive for them, especially to manage differences within student groups.

Insights from educators of programming

Textual programming skills are often difficult to acquire (and teach). In this respect, we gained insights from the participating programming teachers: (a) Using hybrid programming environments can benefit students when learning textual programming. In our case, the hybrid programming tool demonstrated its utility. Students reported they enjoyed the support of the visual blocks to get to know textual programming. (b) DPL-tools can support teachers who must teach heterogeneous student groups. However, based on discussions with teachers and software developers, it was found that there are only a few DPL-tools available in the context of programming. In anticipation of these tools, we will continue to provide adaptivity using learning management systems, as we did with the i-Learn platform. As the DPL-track demonstrated, the diversity of tools and DPL can work in combination to ensure engagement and appropriate challenges for students with different levels of knowledge.

Next Steps

Looking forward to the next iteration, we have set up an additional follow-up study comprising a pedagogical intervention regarding the implementation of the DPL track in different classroom settings (Van Schoors et al., 2023b). In doing so, we wanted to explore and reflect upon teachers' behaviour and actions while using the DPL track. The findings of interviews and observations will act as a new iteration to further evaluate and optimise the DPL track.

ACKNOWLEDGEMENTS

This study was made possible by the i-Learn project (Grant/Award number: AH.2019.051) which aims to make an open portal that supports DPL-tools for Flemish primary and secondary schools. We thank FTRPRF, Minecraft Education Edition, and Dodona for the support during this study. We also thank the participants who took part in this study. All figures from i-Learn, FTRPRF, Minecraft and Dodona were used with permission.

REFERENCES

Autio, O., Hietanoro, J., & Ruismäki, H. (2011). Taking part in technology education: Elements in students' motivation. *International Journal of Technology and Design Education, 21*(3), 349–361. https://doi.org/10.1007/s10798-010-9124-6

Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM Inroads, 10*(2), 30-36. https://doi.org/10.1145/3324888

Bernacki, M.L., Greene, M.J. & Lobczowski, N.G. (2021). A systematic review of research on personalized learning: Personalized by whom, to what, how, and for what purpose(s*)?. Educational Psychology Review*, *33*(4), 1675–1715. https://doi.org/10.1007/s10648-021-09615-8

BookWidgets. (n.d.). *The perfect content creation tool for teachers in the classroom.* https://www.bookwidgets.com/

Brown, N. C. C., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. *ACM Transactions on Computing Education, 14*(2), 9:1–9:22. https://doi.org/10.1145/2602484

Bruce, K. B. (2018). Five big open questions in computing education. *ACM Inroads, 9*(4), 77–80. https://doi.org/10.1145/3230697

Bunting, L., af Segerstad, Y. H., & Barendregt, W. (2021). Swedish teachers' views on the use of personalised learning technologies for teaching children reading in the English classroom. *International Journal of Child-Computer Interaction*, 27, 100236. https://doi.org/10.1016/j.ijcci.2020.100236

Capovilla, D., Berges, M., Mühling, A., & Hubwieser, P. (2015). Handling heterogeneity in programming courses for freshmen. *2015 International Conference on Learning and Teaching in Computing and Engineering*, 197-203. https://doi.org/10.1109/LaTiCE.2015.18

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: A comparison of graphical and textual languages. *Computer Science Education, 29*(1), 23–48. https://doi.org/10.1080/08993408.2018.1547564

Dimitriadis, Y., Martínez-Maldonado, R., & Wiley, K. (2021). Human-centered design principles for actionable learning analytics. In T. Tsiatsos, S. Demetriadis, A. Mikropoulos, & V. Dagdilelis (Eds.), *Research on E-Learning and ICT in Education: Technological, Pedagogical and Instructional Perspectives* (pp. 277–296). Springer International Publishing. https://doi.org/10.1007/978-3-030-64363-8_15

Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., Joshi, D., Robinson, R., & Andrews, A. (2019). PRADA: A practical model for integrating computational thinking in K-12 education. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 906–912. https://doi.org/10.1145/3287324.3287431

Elen, J. (2020). "Instructional disobedience": A largely neglected phenomenon deserving more systematic research attention. *Educational Technology Research and Development, 68*(5), 2021-2032. https://doi.org/10.1007/s11423-020-09776-3

Falkner, K., Vivian, R., & Falkner, N. (2015, January). Teaching computational thinking in k-6: The cser digital technologies mooc. *Proceedings of the 17th Australasian computing education conference.* 160, 63-72. https://www.researchgate.net/publication/277022717_Teaching_Computational_Thinking_in_K-6_The_CSER_Digital_Technologies_MOOC

FitzGerald, E., Kucirkova, N., Jones, A., Cross, S., Ferguson, R., Herodotou, C., Hillaire, G., & Scanlon, E. (2018). Dimensions of personalisation in technology-enhanced learning: A framework and implications for design: Dimensions of personalisation in TEL. *British Journal of Educational Technology, 49*(1), 165–181. https://doi.org/10.1111/bjet.12534

Flemish Government. (2019). *Educational goals: Secondary education - 1st grade - after modernization. Educational Goals - Outcomes.* https://onderwijsdoelen.be/modernisatie?onderwijsstructuur=SO_1STE_GRAAD

Flemish Parliament. (2018). *Draft decree: on educational objectives for the first grade of secondary education.* http://docs.vlaamsparlement.be/pfile?id=1430279

Gomes, A. J., Santos, A. N., & Mendes, A. J. (2012). A study on students' behaviours and attitudes towards learning to program. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 132-137. https://doi.org/10.1145/2325296.2325331

Govender, I. (2006). *Learning to program, learning to teach programming: Pre- and in-service teachers' experiences of an object-oriented language* [Unpublished doctoral dissertation]. University of South Africa. http://hdl.handle.net/10500/1495

Groff, J. (2017). *Personalized learning: The state of the field & future directions*. Center for Curriculum Redesign. https://curriculumredesign.org/wp-content/uploads/PersonalizedLearning_CCR_April2017.pdf

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher,42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Holmes, W., Anastopoulou, S., Schaumburg, H., & Mavrikis, M. (2018). *Technology-enhanced personalised learning: Untangling the evidence.* Robert Bosch Stiftung GmbH, Stuttgart. http://www.studie-personalisiertes-lernen.de/en/

Homer, M., & Noble, J. (2017). Lessons in combining block-based and textual programming. *Journal of Visual Languages and Sentient Systems, 3*(1), 22–39. https://doi.org/10.18293/VLSS2017-007

Kuhn, J. (2018). Minecraft: Education Edition. *CALICO Journal, 35*(2), 214–223. https://doi.org/10.1558/cj.34600

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37. https://doi.org/10.1145/1929887.1929902

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Major, L., & Francis, G. A. (2020). *Technology-Supported Personalised Learning: A Rapid Evidence Review*. Zenodo. https://doi.org/10.5281/ZENODO.4556925

McKenney, S. E., & Reeves, T. C. (2012). *Conducting educational design research*. Routledge.

McKenney, S., & Reeves, T. C. (2014). Educational design research. In J. Spector, M. Merril, J. Elen, & M. Bishop (Eds.), *Handbook of research on educational communications and technology* (pp. 131-140). Springer.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing, 1*(1), 97–123. https://doi.org/10.1016/S1045-926X(05)80036-9

Noone, M., & Mooney, A. (2018). Visual and textual programming languages: A systematic review of the literature. *Journal of Computers in Education, 5*(2), 149–174. https://doi.org/10.1007/s40692-018-0101-5

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* Basic Books.

Robinson, W. (2016). From Scratch to Patch: Easing the blocks-text transition. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, 96–99. https://doi.org/10.1145/2978249.2978265

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in Education, 10*(1), 73-88. https://www.ceeol.com/search/article-detail?id=69618

Salac, J., Thomas, C., Butler, C., & Franklin, D. (2021, March). Supporting diverse learners in K-8 computational thinking with TIPP&SEE. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 246-252. https://doi.org/10.1145/3408877.3432366

Sands, P., Yadav, A., & Good, J. (2018) Computational thinking in K-12: In-service teacher perceptions of computational thinking. In M.S. Knine (Ed.), *Computational thinking in the STEM disciplines* (pp. 151-164). Springer.

Tabak, I. (2004). Synergy: A complement to emerging patterns of distributed scaffolding. *Journal of the Learning Sciences, 13*(3), 305–335. https://doi.org/10.1207/s15327809jls1303_3

Tedre, M., & Denning, P. J. (2021). Computational Thinking: A Professional and Historical Perspective. In *Computational Thinking in Education* (pp. 1-17). Routledge.

Tóth, T., & Lovászová, G. (2018). On Difficulties with Knowledge Transfer from Visual to Textual Programming. *DIVAI 2018*, 379-386. https://conferences.ukf.sk/index.php/divai/divai2018/paper/view/2465

Tsai, C. Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior,* 95, 224-232. https://doi.org/ 10.1016/j.chb.2018.11.038

Van den Akker, J. (1999). Principles and methods of development research. In J. van den Akker, R. M. Branch, K. Gustafson, N. Nieveen, & T. Plomp (Red.), *Design Approaches and Tools in Education and Training* (pp. 1–14). Springer. https://doi.org/10.1007/978-94-011-4255-7_1

Van Petegem, C., Deconinck, L., Mourisse, D., Maertens, R., Strijbol, N., Dhoedt, B., De Wever, B., Dawyndt, P., & Mesuere, B. (2022). Pass/fail prediction in programming courses. *Journal of Educational Computing Research*. https://doi.org/10.1177/07356331221085595

Van Schoors, R., Elen, J., Raes, A., & Depaepe, F. (2021). An overview of 25 years of research on digital personalised learning in primary and secondary education: A systematic review of conceptual and methodological trends. *British Journal of Educational Technology, 52*(5), 1798-1822. https://doi.org/10.1111/bjet.13148

Van Schoors, R., Elen, J., Raes, A., Vanbecelaere, S., & Depaepe, F. (2023a). The Charm or Chasm of Digital Personalized Learning in Education: Teachers' Reported Use, Perceptions and Expectations. *TechTrends, 67*(2), 315-330. https://doi.org/10.1007/s11528-022-00802-0

Van Schoors, R., Elen, J., Raes, A., & Depaepe, F. (2023b). Tinkering the Teacher–Technology Nexus: The Case of Teacher-and Technology-Driven Personalisation. *Education Sciences, 13*(4), 349. https://doi.org/10.3390/educsci13040349

Vandewaetere, M., & Clarebout, G. (2014). Advanced technologies for personalized learning, instruction, and performance. In J. M. Spector, M. D. Merrill, J. Elen, & M. J. Bishop (Red.), *Handbook of Research on Educational Communications and Technology* (pp. 425–437). Springer. https://doi.org/10.1007/978-1-4614-3185-5_34

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35. https://dl.acm.org/doi/fullHtml/10.1145/1118178.1118215?casa_token=f6WtosMl8VoAAAAA:ZMhHdUvKpjLF-1oFDwMSOD-Xz89DPQiZGo4yXZfYc4Ri2FIqcpqUYZatexfpGZuGvPE5KhWokd8twb0

Xie, H., Chu, H. C., Hwang, G. J., & Wang, C. C. (2019). Trends and development in technology-enhanced adaptive/personalized learning: A systematic review of journal publications from 2007 to 2017. *Computers & Education, 140,* 103599. https://doi.org/10.1016/j.compedu.2019.103599

APPENDIX A

Overview of discussion topics for students

Topic	Exemplary questions
(1) General	How did you like the learning trajectory? Are some things left unclear? Did you notice mistakes? Would you do things differently?
(2) Learning content	What did you think of the explanations of concepts such as decomposition, algorithmic thinking
(3) Tasks	What did you think about the tasks? Were they too easy/hard? Did you enjoy solving them? Did you expect other tasks?
(4) Clarity of learning goals	What could be the main purpose of this learning trajectory? In what way do these tools contribute to your learning?
(5) Support/instructions	What did you think about the supporting videos/tips/solutions during the learning trajectory? How did you use them?
(6) Evaluation	What is your opinion about the beginning test? What did you think about keymoments A/B/C?

Overview of discussion topics for teachers

Topic	Exemplary questions
(1) General	How did you like the learning trajectory? What do you think about the length? Did you notice mistakes? Would you do things differently? How do you think students will like it?
(2) Theory	What did you think of the theory we provide to the students? Is it broad enough? Would you add/remove certain concepts? Is there a good sequence of theory?
(3) Tasks	What did you think about the tasks in general? Were they too easy/hard? Did you expect other tasks? Would you use a different didactical approach to teach programming?
(4) Clarity of learning goals	Do you think the main purpose of this learning trajectory is clear for the students? Would you change anything with this respect?
(5) Support/instructions	What did you think about the supporting videos/tips/solutions during the learning trajectory?
(6) Evaluation	What is your opinion about the beginning test? What did you think about keymoments A/B/C? Would you add another evaluation moment on any given moment in the learning track?
(7) Adaptivity	How do you like the built-in-adaptivity? Is it sufficient? Would you change anything about the keymoments to influence the adaptivity? Would you increase adaptivity?

APPENDIX B

Overview plans for refinement based on teachers' and students' feedback.

Topic	Teacher Insights	Plans for refinements and design dilemmas
Theory and learning content	Selection of theory was positively regarded, but perceptions of low- and high-skilled teachers differed. Students reported the need for more visuals reinforcing abstract concepts. At the blended and textual programming levels, more explanation and repetition are needed.	- More balance between textual and visual support/instruction for practice and instruction. - More repetition and clarification of theoretical concepts, especially in Phase 2 and 3.
Tasks	Teachers generally thought tasks were sufficient, however, some teachers differed in their suggestions depending on their skill level. Students found textual programming tasks challenging.	- More balance between textual and visual support/instruction for practice and instruction.
Clarity of learning goals	Teachers thought that learning objectives were clear but would like repetitional summaries of goals during the track for further clarity. The learning goals had to be clarified for the students as they were rather unaware.	- clarification of learning goals. A brief overview of (1) explicit learning goals, (2) upcoming software tools and (3) their added value will be added at the beginning and ending of the DPL-track.
Support/instructions	Teachers were unsure about solutions throughout the learning track. They also expected a lot of technical issues due to the use of different tools. Additionally, they were worried about 'gaming the system'. Students discussed 'freedom of choice'.	- Students are expected to alternate between different screens/tabs. This can be a problem for students with, for example, low ICT-skills. More technical support will be provided during transitions.
Evaluation	Teachers would like a call-back moment with regard to quiz questions of key moment A, as well as a final evaluative moment as an end evaluation. Evaluation accurately sorted students into skill groups. Students also felt positively about the evaluations.	- Currently, there is a missed learning opportunity, as students are not provided with answers or feedback when answering quiz questions. This was a deliberate design choice, as feedback (alongside corresponding scores) could hamper the unobtrusive guidance to a personalised DPL-track and therefore also demotivate beginning students with no programming knowledge. An effort will be made to insert some 'recall' learning moments (that also act as feedback) throughout the DPL-track.
Adaptivity	Adaptivity was lauded by all teachers, however, teachers also asked for system-controlled adaptive moments after learner-controlled moments.	- two additional key moments (one after key moment B and one after key moment C; see figures 21 and 22) were added to check whether students who do not wish to receive extra information or tasks, really understand the learning content. If not, they will also go through extra instruction and tasks. Thus, these additional key moments will contain system-controlled adaptivity based on cognitive learner parameters. In doing so, we will create two shared-controlled adaptivity sets.