Paper accepted as: Mørch, A.I., Andersen, R. (2023). Programming with Minecraft Bedrock Up: Modeling, Coding, and Computational Concepts. In: Spano, L.D., Schmidt, A., Santoro, C., Stumpf, S. (eds) End-User Development. IS-EUD 2023. Lecture Notes in Computer Science, vol 13917. Springer, Cham. https://doi.org/10.1007/978-3-031-34433-6_15

Programming with Minecraft Bedrock Up: Modeling, Coding, and Computational Concepts

Anders I. Mørch[1][0000-0002-1470-5234] and Renate Andersen[2][0000-0002-1206-2140]

[1] Department of Education, University of Oslo, Norway
[2] Faculty of Education and International Studies, Oslo Metropolitan University, Norway
andersm@uio.no renatea@oslomet.no

Abstract. As society gradually digitizes, the need increases for computational literacy education or the inclusion of programming and computational thinking in school curricula and teaching practices. A challenge is to reach students with little or no experience in programming but who have played digital games like Minecraft (i.e., the sandbox game). Toward that end we have developed a new teaching method, which guides students from concrete to abstract programming activities stepwise: 1) block building and modeling, 2) high-level functions, and 3) general purpose programming. The method is based on a theoretical framework Action–Breakdown–Repair and is the result of the first iteration of a design-based research (DBR) process that aims to adapt Minecraft Education and its embedded block-based programming language MakeCode to teaching introductory computer science. We demonstrate the method and report the strengths and weaknesses of the first DBR iteration, including high engagement, different levels of abstraction, challenges of understanding computational concepts in working code, and the role of a shared referent in the game world to coordinate program understanding.

Keywords: Minecraft, Programming, Modeling, Computational Thinking, Teaching Method, Action-Breakdown-Repair, Design-Based Research, End-User Development, Block-Based Programming, Levels of Abstraction.

1 Introduction

The main goal of our research is to develop a method of teaching programming to students who are not primarily interested in technology but who need to understand how technology impacts society and educational processes. The rationale for the method is the gradual increase of digitalization in society, most recently seen by generative AI tools, requiring understanding of algorithms. Furthermore, the method must start with concrete activities of a practice situation, in our case a game environment, providing an engaging entry to computer science (CS) for non-CS majors. The method is part of a design-based research (DBR) process, whereby aspects of theory, design, and evaluation develop iteratively [1, 2, 3].

The other rationale for the method and the DBR process is a theoretical framework and pedagogical model, levels of abstraction and ABR model. The notion of levels of

abstraction implies two assumptions. First, to modify an application, an end-user developer must only increase their knowledge proportionally to the complexity of the modification [4]. Second, information presented at one level of abstraction must be more than the sum of the information presented at a lower level [5]. Therefore, we investigated the following abstraction levels in this study: 1) designing with building blocks, 2) building with code blocks (high-level functions), and 3) general-purpose programming and computational concepts. These levels support learners' gradual adoption of more general programming constructs.

The action–breakdown–repair (ABR) model [6, 7, 8] is a three-step process. First, the designers (here, a group of learners) create domain-specific artifacts (*action*). The design activity stops when the designers encounter a problem (*breakdown*). To continue, designers must *repair* the breakdown, and this provides opportunities for learning and teaching. Mørch [9] proposed ABR as a pedagogical model for technology use in education to bridge school learning (formal) and out-of-school learning (practical). In our work, ABR informed our teaching method, whereby actions in the game environment can lead to new understanding by shifting to another, more general, level of programming caused by a breakdown (initiated by human teacher or computer) to enable students to reflect on their activity and modify the game experience by programming at different levels of abstraction (repair).

2 Related Work

2.1 Introducing Programming and CT Using End-User Development

In the first course in programming, students are introduced to the fundamental concepts and techniques of computer programming, such as data types, control structures, and algorithms. Second, students learn how to think similarly to a programmer, referred to as computational thinking (CT) [10, 11, 12, 13]. CT was revived in CS education after Wing's more focused redefinition of CT as a generic skill set everyone must have, naming key concepts such as abstraction, algorithmic design, decomposition, iteration, pattern recognition, and problem orientation [11]. Teachers in our country (Norway) are expected to teach CT in schools in some subjects [14], although no clear instructions or competence connect to how to do this. Accordingly, more research is needed to examine the implications of this in practice.

End-user development (EUD) is a method of software development in which end users create or modify software applications to meet their needs or preferences without the assistance of professional programmers [15, 8, 5, 16, 17]. The connection between CT and end- EUD has been proposed in previous work, but only a few studies have been reported [18, 19]. CT has been suggested a topic in collaborative learning [20], but there is plenty of room for more research.

2.2 Minecraft Studies

Minecraft Education (ME) is an adaptation of the popular game Minecraft for educational purposes [21], providing a digital environment where students can develop both

generic and domain-specific competencies [22, 23]. ME has been found to support engagement, collaboration, the creation of authentic learning activities, and the attainment of learning outcomes [24], as well as teamwork, computer, and coding skills [25]. Studies have highlighted the game's affordances for collaboration in the open-world multiplayer game environment and for developing spatial skills by creating and interacting with 3D objects and scenarios [26]. Minecraft can be used to teach various subjects such as natural science, math, social sciences, language arts, and composition classes [27, 28]. While research on Minecraft for teaching programming is sparse, recent studies have explored its usefulness for teaching programming and coding concepts [29, 30, 31], making use of its embedded programming languages (MakeCode, JavaScript and Python) and EUD techniques. ME provides a technical framework for a gradual transition into programming and users can easily access the programming environment from the game world by direct activation [16, 32] (Figure 1).

Fig. 1. The CodeBuilder (right) is accessed by command "C" from the game environment (left).

2.3 Teaching Methods

Three popular methods for teaching block-based programming are as follows: Use-Modify-Create (UMC), Predict-Run-Investigate-Modify-Make (PRIMM), and Parson's problems. UMC is a framework used to scaffold children's learning in three stages that define the progression of students' programming toward computational thinking [33]. PRIMM is related to UMC but extends it and emphasizes the importance of understanding existing code before changing or modifying it and organizing work around collaborative learning by reading and discussing code to create a shared understanding of how the code works [34]. Instead of starting from an existing program, Parson's problem is a way of learning to program that starts with the correct building blocks (analogous to puzzle pieces to solve a jigsaw puzzle). However, the students,

themselves, must assemble the building blocks. This method aims to teach students the relationships of program constructs [35].

3 Research Method

3.1 Research Question

Based on the related work we found that further research is needed to develop a method for teaching introductory programming that is informed by EUD research and theory, and we ask: How can Minecraft be used as a EUD environment in an educational context to facilitate and motivate the learning of introductory programming, which we address in the remainder of the paper.

3.2 Research Design and Data Collection

We applied a design-based research methodology (DBR). DBR is an interventionist research method that aims to implement changes in a real-world context by combining theory building and practical action [1, 2, 3]. We used DBR to develop a teaching method for use in an educational setting. We chose McKinney and Reeves's [36] "generic model for educational research," since it is an iterative and flexible research method that guides the early stages of the DBR process, which fits our setting. The DBR model consists of three phases: 1) analysis and exploration, 2) design and construction, and 3) evaluation and reflection.

We collected data from students using a questionnaire and observation of the classroom activities. The Minecraft course lasted four weeks with one 2-hour meeting per week. It was one of two seminars that comprised the practical module of a Learning, Design, and Technology course for 2nd year BA students majoring in education at a large public university in Scandinavia. We are in the early phases of our research project and have completed one full DBR cycle (16 participants; 12 questionnaire respondents). Several DBR cycles and iterations may be needed to harness the teaching method. The responses were thematically categorized using aspects of Clarke & Braun's method [37] and combining inductive (data driven) and deductive (theory driven) coding. In the next section, we provide a demonstration of the teaching method as we used it.

4 Demonstration of Teaching Method

In Minecraft, the primary aim of players is to build and interact with visual artifacts and communicate with other players by chat. In this case, the students are given the task of designing a dream house in steps that require programming and traversing levels of abstraction. We investigated the following abstraction levels: 1) designing with building blocks, 2) building with code blocks, and 3) general programming and computational concepts. These levels aim for a gradual transition into programming according to the theoretical framework we presented in Section 1. We show examples of each of these levels through a scenario based on observational data and reproduced snapshots.

4.1 Designing with Building Blocks

The first task took place on Day 1 and was to create a house based on a visual model (picture) provided by the teacher (Figure 2a–b) and a set of building blocks (Figure 2c). The students worked in groups of four and collaborated to solve the task in a shared ME world hosted on the teacher's computer. ME worlds can accommodate up to 30 simultaneous users, and an avatar represents each user (Figure 2a). The students completed the task in about an hour. The group work included gaining a shared understanding of the overall assignment, the first task, and dividing the work into subtasks.

Fig. 2. a) House with an avatar who built the house by manual operations, b) same house seen from another angle; it has two windows and an entrance, and c) building blocks inventory and toolbar with four active blocks: oak planks, spruce planks, air, and glass.

Figure 2 (a–b) shows the model house that the students created in Minecraft, and the building blocks are shown in (c). None of the students found this task difficult, and all enjoyed the experience of designing and modeling, which they called "doing digital Lego." The transfer to programming was not obvious at this stage, as students did not encounter problems other than the time it took to stack blocks manually, which some considered tedious or boring. They were told at the end of the first day that block-based programming is analogous to a builder who calls out basic operations (place and break) to an interpreter who stacks blocks automatically to create visual structures by composition, such as a mason stacks bricks to build real houses. Furthermore, they were told that code blocks are a special type of building block that can be used to create computational structures by composition.

4.2 Building with Code Blocks

On the second day, one week later, the students were asked to recreate the house they had previously built manually—this time, using code blocks to automate construction. Each group hosted its own world so they could work outside class hours. This task was more complicated, and the teacher simplified it thus: 1) using higher-level building blocks, and 2) the blocks were given and inspired by Parson's method [35]. The code blocks are 3D graphic functions ("fill with" and "place" in Figure 3) requiring parameters (x, y, z) and the blocks must be put in the right sequence (Figure 3-right). To simplify the task, the students were given code blocks in a PDF document but scrambled like a jigsaw puzzle (Figure 3-left).

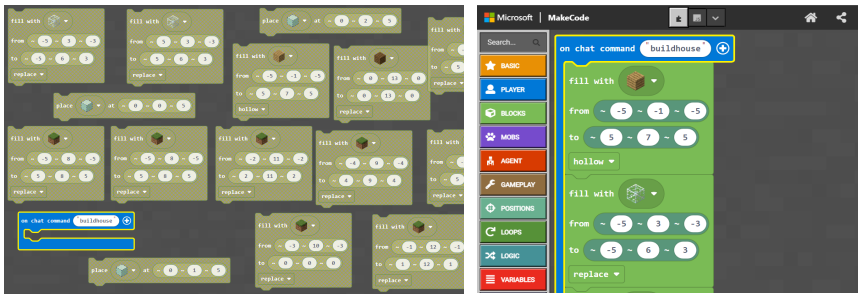

Fig. 3. Left: the students were given 16 code blocks on the second day, which when placed in the right sequence inside the "buildhouse" command block recreated the house shown in Figure 2. The task was adapted from a YouTube video [38].

The students noticed that the blocks could be placed in many different sequences within the chat-command block to solve the task. They also noticed distinctions when they executed the "buildhouse" command, and the parameters (often inadvertently) varied from those in the given code. The teacher's on-demand scaffolding addressed these problems, suggesting *sequences* and *variables* as useful programming concepts. These topics would recur at the beginning of the next meeting in a lecture. Students gained practical experience in the game world to address unexpected situations, such as breaking out of the entrapment on their own or getting help from peers. The students could monitor their progress by comparing their results with the model house, which gave them great pleasure upon (and some frustration about) the task's accomplishment. All students completed the task and took screenshots to be part of the final report.

4.3 General Programming and Computational Concepts

In the third meeting (Day 3), the students were given the task of extending the model house toward the "dream house" that each group documented with a photo or visual image they brought to class. To complete the task, they were allowed to use higher-level code blocks, like Day 2. However, they also had to use at least one loop and one variable, and the loop preferably included a Boolean variable. Before they started, they were given a lecture on the relevant programming concepts and a few related topics with examples (e.g., how to automate the construction of a wall with two loops).

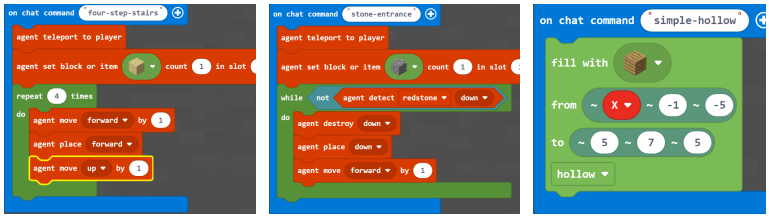

Fig. 4. From left to right: a) loop with four iterations to create stair-like path to 2nd floor, b) loop with conditional test to create cobble stoned path to house that stops when Redstone is underneath the agent, and c) variable that represents the x-value of a hollow cube.

Figure 4 depicts three examples of the kind of code the groups created: a) stairs to the second floor, b) a cobblestone pathway to the front entrance that checks for the appearance of Redstone in the ground, and c) an argument for the build-house command that places the house x meters from the avatar (~x). This task was perceived as the most complex; the students preferred to use the more specific code blocks they had used before. Scaffolding at this stage included bringing to the students' attention problems they may not have discovered independently: 1) efficient code, and 2) general concepts.

The students presented their work to the whole class on Day 4 and wrote a report afterwards. The report asked about reflections on the experiences gained from the seminar, the extent to which the beginning activity of designing and modeling was useful, and the role of higher-level (CT) concepts in conceptualizing the programming activity.

5 Preliminary Results

We organized the empirical findings in terms of strengths and weaknesses, which were the high-level thematic codes. The first iteration of the DBR process had aims of exploration and problem finding. We summarize findings below and then elaborate them.

Strengths: Students said that the levels of abstraction (operationalized as three tasks) provided a gentle introduction to programming: everyone found the first level (modeling) easy and highly engaging. Coding felt progressively more demanding but using specialized code blocks (ME building functions) were easier than general-purpose programming blocks. Everyone collaborated when they built. Several cooperated when they programmed, but not all. Those who worked alone received more help from the teacher. Everyone said they learned something about basic programming concepts. Loops were mentioned by everyone; variables and the Minecraft agent (coding assistant) were also mentioned by several.

Shortcomings: The students could traverse all the levels, but further into the process, they lost an understanding of what they were doing. One student said, "Minecraft is a gentle way to learn programming, but I don't seem to have learned the depth of it, though." Having a shared referent (a common object of understanding) outside the computing domain (in the game world) helped as we elaborate below. The connection between visual structures in Minecraft (building and modeling) and general concepts in programming was difficult to understand despite visual similarity. Everyone found the coordinate system demanding; they struggled to place the blocks correctly when programming. No one mentioned the general (CT) concepts, such as iteration and abstraction, when asked to name what they had learned.

6 Discussion and Conclusions

In this paper, we addressed the following research question: How can Minecraft be used as a EUD environment in an educational context to facilitate and motivate the learning of introductory programming? Our main finding is a novel teaching method that enables non-programmers a gentle introduction to learning programming, which is characterized by three steps: 1) designing with building blocks, 2) building with code blocks,

and 3) general programming and CT. Our method has some similarities to existing teaching methods with block-based programming. In step 2, where the students are given code blocks that they must assemble into a working code [35]. In addition, the students do not start programming from scratch; they are given an external object (model house in the game world) to reconstruct, first by building blocks and then by code blocks, inspired by modeling practice in CS education [39] applied to visual design. Finally, the students created something on their own by choosing their own object: a dream house. The method has similarities to UMC [33] and PRIMM [34], as these methods emphasize reusing and modifying program code before the learners create something new. However, in our method, adaptation (modeling) is a new activity.

Out theoretical framework informed the teaching method by a stepwise process in terms of what to do on each level and the conditions for transferring to the next. This framework emphasizes knoweldge and program modification as two objects that must be coordinated, and levels of abstractions for each software object that lead students into more advanced computational concepts and programming practices upon transfer.

We identified two dilemmas in our findings. First, whereas the students could traverse the three levels and complete the tasks, they could not always explain the program code they had created, which they found frustrating. One the one hand, the shared referent in the game world (model object) helped the learners to create a shared object of understanding that evolved. On the other hand, this object did not evolve to align with the technical (software) object the students created in the CodeBuilder. A cause of some disturbance was the Minecraft coordinate system. The students struggled to understand that variables (e.g., x, y, z) could be used as placeholders for constants (e.g., -5, 3, -3) and more generally arguments to use as input to the buildhouse command.

The other dilemma we identified was that the connection between visual structures in Minecraft and general programming constructs (including CT concepts) was difficult for the students to understand. On the one hand, visual artifacts (building blocks and code blocks) reveal high resemblance by following a similar composition logic based on the jigsaw puzzle metaphor and snapping. On the other hand, connecting code blocks by their names does not follow the same intuitive logic. There is no natural transition from “fill with” to repeat-loop and from integer values to variables. Intermediate levels of abstraction or increased scaffolding may be needed to make verbal transitioning more meaningful.

In summary: Our main idea is that by integrating the learning of programming in Minecraft, students can learn advanced programming concepts without being aware that they are learning them. Based on feedback from the students and evaluations of the method, we found that students could traverse all the levels, but further into the process, they lost understanding of what they were doing, or rather they did not achieve shared conceptual understanding. To address the issues and dilemmas and looking ahead, we suggest that more teaching and scaffolding are needed to harness the object of understanding toward a shared knoweldge object of multiple levels, on the one hand, and to make the general (e.g., CT) concepts applicable to students’ concrete learning activities, on the other. Future work will address the shortcomings by adding another day to the course (for teaching and practice) and designing non-player characters (NPCs) in the game world for automated scaffolding. This requires another cycle of the DBR process.

References

1. Brown, A. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. The Journal of Learning Sciences **2**(2), 141-178 (1992).
2. Collins, A., Joseph, D., Bielaczyc, K.: Design research: Theoretical and methodological issues. Journal of the Learning Sciences **13**(1), 15–42 (2004).
3. Hoadley, C., Campos, F.C.: Design-based research: What it is and why it matters to studying online learning. Educational Psychologist **57**(3), 207–220 (2022).
4. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-tailorable systems: Pressing the issues with buttons. In: Proceedings of CHI'90, pp. 175–182. ACM, New York, NY (1990).
5. Mørch, A.: Three levels of end-user tailoring: Customization, integration, and extension. In: Kyng, M., Mathiassen, L. (eds.) Computers and design in context, pp. 51–76. The MIT Press, Cambridge, MA (1997).
6. Schön, D.A. The reflective practitioner: How professionals think in action. Basic Books, New York (1983).
7. Ehn, P.: Work-oriented design of computer artifacts. Arbetslivscentrum, Stockholm (1988).
8. Fischer, G.: Domain-oriented design environments. Autom. Softw. Eng. **1**(2), 177–203 (1994)
9. Mørch, A.I.: Two 3D virtual worlds as domain-oriented design environments: Closing the educational gap with the action-breakdown-repair model. International Journal of Information and Learning Technology **37**(5), 295–307 (2020).
10. Papert, S.: Mindstorms: Children, computers, and powerful ideas. Basic Books, New York (1980).
11. Wing, J.M. Computational thinking. Commun. ACM **49**(3), 33–35 (2006).
12. Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., Wilensky, U.: Defining computational thinking for mathematics and science classrooms. Journal of Science Education and Technology **25**(1), 127–147 (2016).
13. Shute, V. J., Sun, C., Asbell-Clarke, J.: Demystifying computational thinking. Educational Research Review **22**, 142–158 (2017).
14. Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M. A., Jasutė, E., Malagoli, C., Masiulionytė-Dagienė, V., Stupurienė, G.: Reviewing computational thinking in compulsory education. Report no. JRC128347. Publications Office of the European Union (2022).
15. Fischer, G., Girgensohn, A.: End-user modifiability in design environments. In: Proceedings CHI'90, pp. 183–192. ACM, New York (1990).
16. Wulf, V., Golombek, B.: Direct activation. A concept to encourage tailoring activities. Behav. Inf. Technol. **20**(4), 249–263 (2001).
17. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A.: End-user development: The software shaping workshop approach. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End-user development. Human-Computer Interaction Series, vol 9. Springer, Dordrecht (2006).
18. Basawapatna, A., Koh, K.H., Repenning, A., Webb, D.C., Marshall, K.S.: Recognizing computational thinking patterns. In: Proceedings of the 42nd ACM technical symposium on computer science education (SIGCSE'11), pp. 245–250. ACM, New York (2011).
19. Barricelli, B.R., Fogli, D., Locoro, A.: EUDability: A new construct at the intersection of end-user development and computational thinking. Journal of Systems and Software 195(c) 111516 (2023).
20. Kafai, Y.B.: From computational thinking to computational participation in K-12 education. Commun. ACM **59**(8), 26–27 (2016).

21. Ellison, T.L., & Evans, J.N.: Minecraft, teachers, parents, and learning: What they need to know and understand. School Community Journal **26**(2), 25–43 (2016).
22. Mørch, A.I., Eie, S., Mifsud, L.: Tradeoffs in combining domain-specific and generic skills' practice in Minecraft in social studies in teacher education. In: Proc. of Fifth International Workshop on Cultures of Participation in the Digital Age, CoPDA 2018, pp. 44–52. Castiglione della Pescaia, Italy (published https://ceur-ws.org/Vol-2101/paper6.pdf) (2018).
23. Rahimi, S., Walker, J.T., Lin-Lipsmeyer, L., Shin, J.: Toward defining and assessing creativity in sandbox games. Creativity Research Journal (2023).
24. Callaghan, N.: Investigating the role of Minecraft in educational learning environments, Educational Media International **53**(4), 244–260 (2016).
25. Karsenti, T., Bugmann, J.: Exploring the educational potential of Minecraft: The case of 118 elementary-school students. In: Proceedings ICEduTech2017, International Conference on Educational Technologies, International Association for Development of the Information Society, Sydney, Australia (2017).
26. Carbonell-Carrera, C., Jaeger, A.J., Saorín, J.L., Melián, D., de la Torre-Cantero, J.: Minecraft as a block-building approach for developing spatial skills. Entertainment Computing **38**, 100427 (2021).
27. Baek, Y., Min, E., Yun, S.: Mining educational implications of Minecraft. Computers in the Schools **37**(1), 1–16 (2020).
28. Andersen, R., Eie, S., Mørch, A.I., Mifsud, L., Rustad, M.: Rebuilding the industrial revolution: Using Minecraft in teacher education in social studies. In: Proceedings of the 15th International Conference of the Learning Sciences (ICLS 2021), pp. 27-34. International Society of the Learning Sciences, Bochum, Germany (2021).
29. Klimová, N., Šajben, J., Lovászová, G.: Online game-based learning through Minecraft: Education edition programming contest. In: 2021 IEEE Global Engineering Education Conference (EDUCON), pp. 1660–1668. IEEE Press, Washington, DC (2021).
30. Bile, A.: Development of intellectual and scientific abilities through game-programming in Minecraft. Education and Information Technologies, 27(5), 7241–7256 (2022).
31. Kutay, E., Oner, D.: Coding with Minecraft: The development of middle school students' computational thinking. ACM Transactions on Computing Education **22**(2), 1–19 (2022).
32. Mørch, A.I.: Aspect-oriented software components. In Patel, N. (ed.) Adaptive evolutionary information systems, pp. 105–23. Idea Group Publishing, Hershey, PA (2003).
33. Lee, I., Martin, F., Denner, J, Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L.: Computational thinking for youth in practice. ACM Inroads **2**(1), 32–37 (2011)
34. Sentance, S., Waite, J., Kallia, M.: Teaching computer programming with PRIMM: A sociocultural perspective, Computer Science Education **29**(2), 136–176 (2019).
35. Parsons, D., Haden, P.: Parson's programming puzzles: A fun and effective learning tool for first programming courses. In: Proceedings of the 8th Australasian Conference on Computing Education, vol. 52, pp. 157–163. Australian Computer Society, Canberra (2006).
36. McKenney, S., Reeves, T.: Conducting educational design research. 2nd edn. Routledge, Oxford (2019).
37. Clarke, V., Braun, V.: Thematic analysis. In: Michalos, A.C. (eds.) Encyclopedia of quality of life and well-being research. Springer, Dordrecht (2014).
38. Minecraft Education Edition - How to Code a House, https://www.youtube.com/watch?v=APSo9qFngoM, last accessed 2023/4/16.
39. Madsen, O.L., Møller-Pedersen, B.: A unified approach to modeling and programming. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds). Model driven engineering languages and systems (MODELS 2010). LNCS, vol. 6394. Springer, Berlin, Heidelberg (2010).