Logical Minecraft: Teaching Logical Thinking through Games

Simon Vandevelde[1,2,3]

[1]KU Leuven – Department of Computer Science De Nayer Campus, 2860 Sint-Katelijne-Waver Belgium
[2]Leuven.AI – KU Leuven Institute for AI, B-3000 Leuven, Belgium
[3]Flanders Make@KULeuven, Belgium

Abstract
Game-based learning (GBL) is a common approach to teach computational thinking skills to children, as it increases engagement and student motivation. In the literature, the video game Minecraft is steadily gaining popularity as a teaching environment in this context. In this paper, we look at how Minecraft can be used to teach *logical thinking* instead. We outline a few ideas for pedagogical applications within Minecraft, and discuss two of these in detail. In general, we think Minecraft forms an excellent environment for teaching logic, logic programming, and logical thinking, warranting future research on the development of pedagogical material.

Keywords
Logic Programming, Education, Minecraft, Logical Thinking, Game-based Learning

1. Introduction

Logical thinking is an important skill in our everyday life. Being able to correctly and consciously reason over a problem by, e.g., explicitly applying rules to deduce or verify new information, is a cornerstone of many scientific fields, such as mathematics, philosophy, computer science, Artificial Intelligence, and more. Moreover, by training our reasoning skills we become more aware of logical fallacies and reasoning biases. This makes logic and logical reasoning a skill worth sharpening in younger generations.

In recent decennia, we have seen a similar push for the teaching of Computational Thinking (CT) as a skill that transcends disciplines. However, while high-quality material on teaching CT is plentiful, there is an obvious shortage of good material on teaching *logic*, especially when targeting younger age groups. We believe this is due to two main reasons:

1. Logic does not *do* anything. Compared to interactive programming environments such as Scratch, in which players can program small games, pure logic is boring and therefore difficult to focus on.
2. Logic has a steep learning curve, and can be tricky to conceptualize without prior experience. Moreover, logic typically relies on mathematical symbols such as $\forall, \exists, \Rightarrow, \ldots$, which might scare off younger audiences.

These difficulties make designing captivating material on teaching logic and/or logical thinking for K-12 challenging. Still, we have recently seen renewed interest in teaching logic, with works such as [1, 2, 3, 4, 5, 6, 7] presenting various creative methods for doing so.

An important aspect of teaching material in general is whether it succeeds in capturing the attention of children. If the examples and exercises are not sufficiently engaging, they might risk losing interest as soon as the material becomes too difficult. To prevent this loss of attention, teaching through (video) games is an increasingly popular approach. In fact, in the context of CT education for K-12, this is nowadays already quite common.

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026), co-located with the Federated Logic Conference (FLoC 2026), July 18–19 and 24–25, 2026, Lisbon, Portugal.
s.vandevelde@kuleuven.be (S. Vandevelde)
https://simonvandevelde.be (S. Vandevelde)
0000-0001-7312-3675 (S. Vandevelde)

Figure 1: Screenshot of a house in Minecraft, built by the author's 10-year-old niece.

Based on this, it is clear that a game-based learning approach for logic would also help increase engagement. Therefore, in this paper, we present some concrete ideas on how the hit video game *Minecraft*, one of the all-time best-sellers, could be used to teaching logic, logical thinking, and/or logic programming by finding existing Minecraft application of typical logical examples.

2. Minecraft and its Application in Education

Released in 2011, Minecraft is voxel-based *sandbox game* that has taken the world by storm, quickly becoming one of the most played video games in history. In this game, the world is built up of cubes of different types (e.g., stone, dirt, wood, . . .). By "sandbox game" we mean that, like a sandbox at a playground, there is no pre-defined end-goal to be achieved. Players are free to play the game in whatever way they enjoy most, which is typically a mix of exploring the world, collecting various resources, and building constructions (e.g., the house in Fig. 1). The game itself is also highly extensible, featuring an active community of people developing *mods*, which extend Minecraft with custom functionalities. Thanks to this open and diverse environment, Minecraft offers endless fun and forms a great creative outlet for children.

Given the game's popularity and malleability, educators have started studying possible roles for Minecraft as part of *game-based learning* (GBL) [8]. The main idea in GBL is to use games to teach concepts, so that children will feel more motivated, and therefore have a better time engaging with the material. By playing the game, they become completely submerged in the material, and can "learn through osmosis". GBL has a strong history in the pedagogical theory of *constructionism* [9], which emphasizes learning through experience.

According to a recent literature review [10], Minecraft is steadily gaining popularity within education research, showing a rising trend year after year. Indeed, thanks to its "sandbox"-nature, the game lends itself well to teaching a wide variety of concepts. For instance, [11] shows how activities in Minecraft can be used to teach real-life skills; e.g., by analysing the in-game random fishing mechanic, students learn about data-gathering and probabilistic events. Similarly, [12] presents a Minecraft mod to simulate flooding, used to teach about flood risks and flood mitigation strategies in building construction. For more examples, we refer to [10, 13, 14].

Minecraft is also a popular environment within the context of CT education. For instance, [15] introduced a mod to allow programming in-game robots in the Lua language, and [16] created a mod

to program robots through Minecraft's blocks. Both conclude that children thoroughly enjoyed these experiences, leading to an increased interest in programming.

Mojang, the company behind Minecraft, has also published a version specifically intended for the classroom, called *Minecraft: Education Edition* (MEE). As part of MEE, children can learn programming by writing programs directly inside Minecraft, through either the Python programming language, or a drag-and-drop blocks-based language similar to Scratch [17] and Blockly [18]. Children find this way of learning highly enjoyable [19, 20], with the blocks-based formalism particularly successful when targeting a younger learning audience. However, while some research reports that CT concepts can be learned effectively [21, 22], others report a lack of conceptual understanding in the children [23]. In general, literature so far has only tested a limited number of students, and more extensive experiments are required to draw concrete conclusions.

3. How Minecraft can Support Logic

As far as we are aware, there is no similar academic literature on Minecraft for teaching logic, logical thinking, or logic programming. The closest we have been able to find is [14], which briefly explains how children can learn boolean gate logic (AND, OR, XOR, . . .). Still, we feel that Minecraft holds the potential for much more. In what follows, we outline two concrete examples, and briefly touch on a range of others.

3.1. Path Planning

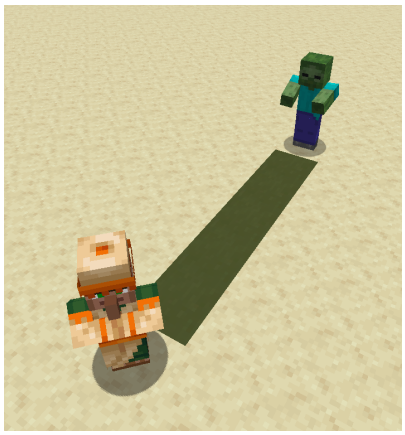
In Minecraft, worlds are populated by both friendly and hostile *non-playable characters*, i.e., entities within the games that the player can interact with. These entities also come with their own behavior, allowing them to react to the player but also to each other. For instance, if a hostile *zombie* character sees a friendly *villager*, it will try to attack it. The zombie can only attack the villager when right next to it, so it must first find a path to reach it, as shown in Fig. 2a. Interestingly, we see similar path finding applications in logic programming as well, for instance in fields such as Multi-Agent Planning Systems [24].

Using the example of zombie pathing, we could develop course material in which we systematically abstract away the details to get to a formal representation of the problem, and solve it using logic programming. Children can be coached to think about this problem themselves by asking the right questions, such as "Standing on block *b*, where can the zombie go to?" and "If he moves to an adjacent block, which blocks are reachable then?". Together with drawn abstractions (e.g. Fig. 2d), they could come up with a definition of reachability in their own words without even realising it. For instance, they might come up with:

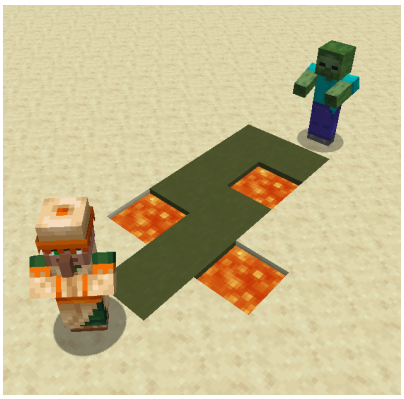
```
A block is reachable if it is next to block b.
A block is reachable if it is next to a block c that is reachable from block b.
```

As a next step, children can formalise this definition in a logic programming language. Depending on their age group, they could write the code directly, or use a more child-friendly "syntax-less front-end" such as a blocks-based notation or a Controlled Natural Language (CNL) [25]. Blocks-based notations for logic (programming), like Blockly Prolog [26], Blawx [27] and similar, have proven to work well for younger children which might struggle with typing or rigid syntaxes [28, 29]. Alternatively, a domain-specific CNL for path finding could also offer a low-barrier formalism for children that can allow expressions closer to the ones they came up with in natural language. Importantly, independent of the formalism, they can always fall back on their knowledge of Minecraft and their drawing as a conceptual model to help in the formalization. A natural Prolog translation for the example sentences can be as follows:

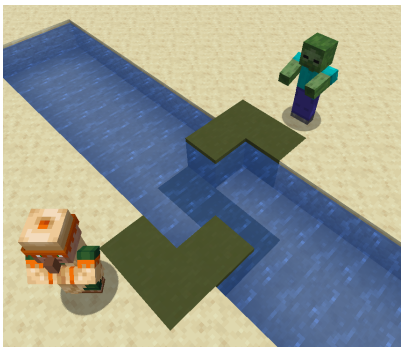
```
reachable(X) :- next_to(b, X)
reachable(X) :- reachable(C), next_to(C, X).
```



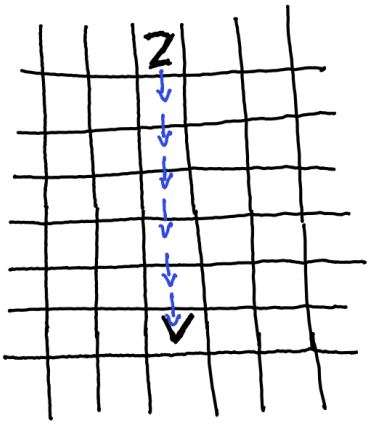
(a) Straight path



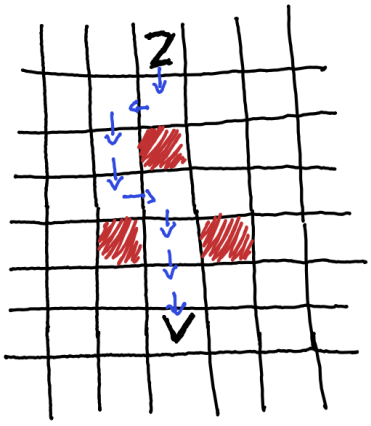
(b) Pathfinding around lava



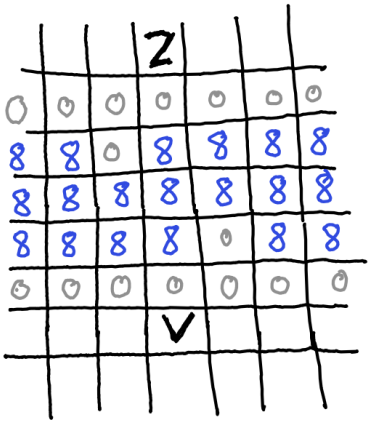
(c) Path finding through water, which has a cost associated to it



(d) Abstract path of Fig. 2a



(e) Abstract path of Fig. 2b



(f) Weight map of Fig. 2c

Figure 2: Pathfinding examples of a *Zombie* (Z) going to a *Villager* (V). Green blocks indicate the Zombie's path.

Using visualisations in Minecraft, or just with pen and paper, children can then verify the correctness of their formalization. This approach of (i) introducing a concept, (ii) abstracting away unneeded details, (iii) expressing it in natural language and (iv) formalizing in a formal languages can be repeated multiple times with slightly increased complexity. For example:

- Expressing the reachability relation
- Finding a path from the zombie to the villager using the reachability relation
- Finding an optimal path (i.e., shortest route)
- Path finding around impossible terrain such as lava (Fig. 2b and Fig. 2e)
- Weighted path finding, like the actual zombie path implementation (Fig. 2c and Fig. 2f).

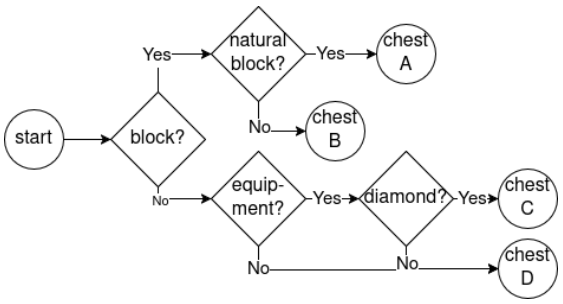
Each step builds further on the previous, but adds additional complexity to challenge the kids and keep them interested. If all goes well, by the end they'll have modelled the path planning as implemented in Minecraft. Moreover, they might be comfortable enough in the problem domain to try to expand upon it, or model the path finding of other entities.

To make learning maximally engaging, we could develop a mod to directly couple the logic program to the in-game zombie behavior. In this way, children can not only come up with pathing behavior, but also see it in action in the digital world.

3.2. Item Sorting

In version 1.21.9, Minecraft added a new non-hostile entity: the *Copper Golem* (Fig. 3a). As players continue playing in a world, they will typically collect a great number of resources, which they will

(a) A Copper Golem has taken feathers out of the copper chest, looking to store it in a normal chest.



(b) Example of an item sorting flowchart.

Figure 3: Copper Golem and its behavior.

safely store away in chests. Copper golems can help to keep chests organized by automatically sorting items placed in their copper chest directly in the players' chests. After a player deposits their items in the copper chest, the golem sorts them using a simple algorithm:

1. Grab item from copper chest
2. Go to a nearby player chest
3. Check if this item is also present
4. If successful, store item and go back to step (1)
5. If unsuccessful, move to the next unvisited chest and go to step (3)

There are a few obvious downsides to this algorithm, as it is essentially a linear search. Most importantly, if the golem is holding an item that does not appear in any of the chests, they will become *stuck*, fruitlessly searching the same chests over and over. This loop is only broken when a player manually takes the item out of the golem's hands and sorts it themselves.

In our opinion, the copper golem offers a great conceptual example for classification using logic (programming), which could teach them how to logically *define* concepts. Children could design their own sorting rules, in which they specify which items are supposed to go in which chests. Instead of enumerating items individually, they group items together in self-defined categories and designate each category a chest. For instance, pickaxes and swords could be grouped in the category *equipment* and stored in chest *A*, whilst stone and wood can be grouped in *building blocks* which are stored in chest *B*.

We envision a two-step approach to these sorting rules. Firstly, the children should come up with a flowchart, using diamond decision shapes to create an overview of which categories are required. Fig. 3b shows an example in which four categories (block, equipment, natural resource, diamond) are used in the sorting flow. Next, they could define each category in natural language, and then formalise them logically in whatever formalism works best. For instance:

```
A block is an item that can be placed.
A natural block is a block that has no crafting recipe.
A piece of equipment is diamond quality when a diamond is used in its crafting recipe.
```

Can be written in pure classical first-order logic as follows:

$$\forall x \in Item : block(x) \Leftrightarrow can\_place(x).$$
$$\forall x \in Item : natural(x) \Leftrightarrow block(x) \wedge \neg(\exists x_2 \in Item : recipe(x, x2)).$$
$$\forall x \in Item : diamond(x) \Leftrightarrow equipment(x) \wedge recipe(x, diamond).$$

Note that item properties, such as *can_place* and *equipment*, would need to be provided up-front as they cannot be trivially defined.

On top of a "free play" environment to toy with sorting rules, we could have the children work through pre-defined exercises as "levels". We envision such a level to consist of a set of input blocks of various types, a sorting algorithm that is hidden from them, and the required outcome of the sorting. The players then need to figure out the sorting algorithm using the approach described above, until theirs reaches the desired outcome. This approach can straightforwardly scale from easy levels to more difficult levels that require many sub-decisions. Along the way, they are learning how to define concepts formally, and learning to think in terms of *flows*.

3.3. Other Examples

As mentioned in the beginning of this section, there are more possible matches between Minecraft and logic (programming) than path planning and item sorting. Indeed, Minecraft's emphasis on creativity allows us seemingly endless possibilities. To motivate this, we briefly introduce a few additional examples:

- Minecraft lends itself well to artistic expression, with many players striving to build houses, landscapes, or other in the most aesthetically pleasing ways. It also has a lively "pixel art" scene, where artists place blocks in a 2D-plane to form images when looked at from afar, similar to how digital images work. This holds a potential synergy with work such as [30], which emphasizes teaching logic programming through visual methods. In this case, Minecraft could be used to visualize generative art, and make it more tangible.
- Configurator tools are typical example applications for logic programming [31]. In Minecraft, we could show this using a "house configurator", in which children can express design rules and constraints over a house. For instance, they could set color schemes using matching blocks, add a constraint on the types of blocks that can be used for flooring, express if they want a flat roof or a saddle roof, etc. In a next step, they can generate a few houses matching their descriptions, and inspect them in their Minecraft worlds. Whenever the houses are not to their liking, they can go back and modify their configuration knowledge.
- ComputerCraft is a Minecraft mod that adds "turtles", which are in-game robots that can be programmed in Lua. Each robot can also be equipped with tools to allow them to interact with the objects and blocks in the world. It seems feasible to extend this mod with support for a logic programming language, such as Prolog or ASP [32]. In this way, players could program agents in their worlds, and use them to automate certain aspects of the game.

In general, it seems that for every aspect or application of logic (programming), one can find an interesting example using concepts from Minecraft. This seems in line with other pedagogical approaches that discuss Minecraft as an environment for game-based learning, making the game well-suited as a general "problem domain".

4. Conclusion

Teaching logical reasoning to children is a worthwhile endeavor, but engaging teaching material can be tough to create. As logic by itself "does nothing", it is important to find interesting application domains to prevent the teaching material from being boring.

In this paper, we present Minecraft as a suitable *game-based learning* environment for logic, logic programming, and/or logical thinking in general. Research shows that game-based learning increases student motivation and therefore engagement with the material. Moreover, thanks to Minecraft's popularity, it is a well-known game for many children already.

As concrete examples, we elaborated on two main ideas: path planning and item sorting. Path planning is a classical application of logic programming, which can straightforwardly be demonstrated

using in-game entities. This gives children a better mental models of what they're supposed to do, and allows them to visualize their results in Minecraft directly. Item sorting through copper golems learns children to think in *flows*, and how to define concepts in terms of others, which are both crucial skills for logical thinking. Besides these two main ideas, we also find that Minecraft's sandbox environment offers many more teaching opportunities, especially when paired with child-friendly logical formalisms such as blocks-based notations or domain-specific CNLs..

In the context of game-based learning, it is also worth stating that Minecraft is not the only game that lends itself well to teaching logical thinking. In general, all games that (a) offer room for creativity, (b) are more or less "sandboxes", and (c) can be extended using mods, could be eligible teaching environments. Some examples of such games include Stardew Valley, Terraria, and Factorio.

As a final conclusion to our paper, we would like to wholeheartedly invite all interested readers to take these ideas and continue to work on them; they are hereby provided free of any intellectual properties. We would be delighted if anyone finds the time and budget to concretely work out the ideas outlined in this text, and transform them into actionable learning material.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] L. A. Cecchi, J. P. Rodríguez, V. Dahl, Logic Programming at Elementary School: Why, What and How Should We Teach Logic Programming to Children?, in: D. S. Warren, V. Dahl, T. Eiter, M. V. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog: The Next 50 Years, Springer Nature Switzerland, Cham, 2023, pp. 131–143. URL: https://doi.org/10.1007/978-3-031-35254-6_11. doi:10.1007/978-3-031-35254-6_11.

[2] J. Archer, R. Eckel, J. Hawkins, J. Wang, D. Musslewhite, Y. Zhang, A Study of Students' Learning of Computing through an LP-Based Integrated Curriculum for Middle Schools, in: EAAI 2023, 2023.

[3] G. Gupta, E. Salazar, J. Arias, Computational thinking with logic programming, in: Proceedings of the 2nd Workshop on Prolog Education (PEG 2024), Association for Logic Programming, Dallas, USA, 2024. URL: https://ceur-ws.org/Vol-3799/paper8PEG2.0.pdf.

[4] K. Reale, Logic and answer set programming in high school: Two learning unit experiences, in: Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming (ICLP-WS-DC 2025), volume 4117 of *CEUR Workshop Proceedings*, CEUR-WS.org, Rende, Italy, 2025. URL: https://ceur-ws.org/Vol-4117/PEG-Short-2.pdf.

[5] A. Platzer, The Significance of Symbolic Logic for Scientific Education, in: E. Sekerinski, L. Ribeiro (Eds.), Formal Methods Teaching, Springer Nature Switzerland, Cham, 2024, pp. 3–22.

[6] J. Arias, Teach the importance of logic (programming) in Computer Science and why it is important, in: 2023 International Symposium on Computers in Education (SIIE), 2023, pp. 1–6. doi:10.1109/SIIE59826.2023.10423714.

[7] Y. Zhang, J. Wang, F. Bolduc, W. G. Murray, W. Staffen, A preliminary report of integrating science and computing teaching using logic programming, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 2019, pp. 9737–9744.

[8] J. L. Plass, B. D. Homer, C. K. Kinzer, Foundations of Game-Based Learning, Educational Psychologist 50 (2015) 258–283. URL: https://www.tandfonline.com/doi/abs/10.1080/00461520.2015.1122533. doi:10.1080/00461520.2015.1122533.

[9] S. A. Papert, Constructionism: A New Opportunity for Elementary Science Education, NSF Award 87 (1987) 51190. URL: https://ui.adsabs.harvard.edu/abs/1987nsf....8751190P.

[10] J. García-Álvarez, J. Acevedo-Borrega, Minecraft as a Pedagogical Tool: A Systematic Literature

Review (2014–2024), Games and Culture (2025) 15554120251341034. URL: https://doi.org/10.1177/15554120251341034. doi:10.1177/15554120251341034.

[11] S. Ivanov, B. Yordanov, Application of minecraft: Education in mathematics and CMIT classes, examples and practices, in: Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024) - Volume 2, SCITEPRESS, Sofia, Bulgaria, 2024, pp. 517–524. URL: https://www.scitepress.org/Papers/2024/126169/126169.pdf. doi:10.5220/0012616900003693.

[12] E. Emiroglu, C. A. Grant, Y. Sermet, I. Demir, Floodcraft: Game-based interactive learning environment using Minecraft for flood mitigation for K-12 education, International Journal of Disaster Risk Reduction 130 (2025) 105799. URL: https://www.sciencedirect.com/science/article/pii/S2212420925006235. doi:10.1016/j.ijdrr.2025.105799.

[13] D. Short, Teaching scientific concepts using a virtual world–minecraft, Teaching Science 58 (2012) 55–58. URL: https://eric.ed.gov/?id=EJ991295.

[14] G. Ekaputra, C. Lim, K. I. Eng, Minecraft: A game as an education and scientific learning tool, in: Proceedings of the Information Systems International Conference (ISICO) 2013, Departemen Sistem Informasi, Institut Teknologi Sepuluh Nopember, Bali, Indonesia, 2013. URL: https://www.researchgate.net/publication/261671901_Minecraft_A_Game_as_an_Education_and_Scientific_Learning_Tool.

[15] B. Wilkinson, N. Williams, P. Armstrong, Improving student understanding, application and synthesis of computer programming concepts with minecraft, in: The European Conference on Technology in the Classroom 2013, The International Academic Forum, 2013.

[16] C. Zorn, C. Wingrave, E. Charbonneau, J. J. J. LaViola, Exploring minecraft as a conduit for increasing interest in programming, in: Proceedings of the Foundations of Digital Games Conference (FDG 2013), Crete, Greece, 2013. URL: https://www.eecs.ucf.edu/~jjl/pubs/paper46_zorn_etal.pdf.

[17] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, Y. Kafai, Scratch: Programming for all, Communications of The Acm 52 (2009) 60–67. URL: https://doi.org/10.1145/1592761.1592779. doi:10.1145/1592761.1592779.

[18] N. Fraser, Ten things we've learned from Blockly, in: 2015 IEEE Blocks and beyond Workshop (Blocks and Beyond), 2015, pp. 49–50. doi:10.1109/BLOCKS.2015.7369000.

[19] P. Voštinár, R. Dobrota, Minecraft as a Tool for Teaching Online Programming, in: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), 2022, pp. 648–653. doi:10.23919/MIPRO55190.2022.9803384.

[20] N. Klimová, J. Šajben, G. Lovászová, Online Game-Based Learning through Minecraft: Education Edition Programming Contest, in: 2021 IEEE Global Engineering Education Conference (EDUCON), 2021, pp. 1660–1668. doi:10.1109/EDUCON46332.2021.9453953.

[21] A. Bile, Development of intellectual and scientific abilities through game-programming in Minecraft, Education and Information Technologies 27 (2022) 7241–7256. URL: https://doi.org/10.1007/s10639-022-10894-z. doi:10.1007/s10639-022-10894-z.

[22] E. Kutay, D. Öner, Coding with minecraft: The development of middle school students' computational thinking, ACM Transactions on Computing Education 22 (2022) 1–19. doi:10.1145/3471573.

[23] A. I. Mørch, R. Andersen, Programming with Minecraft Bedrock Up: Modeling, Coding, and Computational Concepts, in: L. D. Spano, A. Schmidt, C. Santoro, S. Stumpf (Eds.), End-User Development, Springer Nature Switzerland, Cham, 2023, pp. 230–240.

[24] E. Erdem, D. Kisa, U. Oztok, P. Schüller, A general formal framework for pathfinding problems with multiple agents, Proceedings of the AAAI Conference on Artificial Intelligence 27 (2013) 290–296. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8592. doi:10.1609/aaai.v27i1.8592.

[25] T. Kuhn, A Survey and Classification of Controlled Natural Languages, Computational Linguistics 40 (2014) 121–170. URL: https://doi.org/10.1162/COLI_a_00168. doi:10.1162/COLI_a_00168.

[26] N. Holtz, Blockly Prolog, 2021. URL: https://github.com/niklas-holtz/blockly-prolog.

[27] J. Morris, Blawx: Web-based user-friendly rules as code, in: Proceedings of the Workshop on Goal-Directed Execution of Answer Set Programs (GDEASP 2022), CEUR-WS.org, Haifa, Israel,

2022. URL: https://ceur-ws.org/Vol-3193/paper4GDE.pdf.

[28] S. A. Villarroel, C. N. Gimenez, J. P. Rodríguez, L. A. Cecchi, Democratising access to logic programming: A web application design tool for querying prolog code, in: Proceedings of the 2nd Workshop on Prolog Education (PEG 2024), CEUR-WS.org, Dallas, USA, 2024. URL: https://ceur-ws.org/Vol-3799/paper5PEG2.0.pdf.

[29] S. Vandevelde, J. Vennekens, FOLL-E: Teaching First Order Logic to Children, Proceedings of the AAAI Conference on Artificial Intelligence 37 (2023) 15869–15876. doi:10.1609/aaai.v37i13.26884.

[30] C. Jendreiko, The Simple Generative Logic Grammar: A tool for teaching logical thinking through visual research in art and design, in: Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, CEUR-WS.org, 2025, pp. 9–16. URL: https://ceur-ws.org/Vol-4117/PEG-Regular-5.pdf.

[31] J. Baumeister, K. Herud, M. Ostrowski, J. Reutelshöfer, N. Rühling, T. Schaub, P. Wanko, Towards Industrial-Scale Product Configuration, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning, Springer Nature Switzerland, Cham, 2025, pp. 71–84.

[32] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, T. Schaub, ASP-Core-2: Input language format, ASP Standardization Working Group (2012).