

From Blocks to Code: A Game-based Modding Workshop for Bridging Programming Paradigms

André Matutat¹, Benjamin Franke¹, Birgit Christina George¹, Jörg-Michael Keuntje² and Carsten Gips¹

¹Campus Minden, University of Applied Sciences and Arts (HSBI), Germany

²Department of Economics, University of Applied Sciences and Arts (HSBI), Bielefeld, Germany

andre.matutat@hsbi.de

benjamin.franke@hsbi.de

bc.george@hsbi.de

joerg-michael.keuntje@hsbi.de

carsten.gips@hsbi.de

Abstract: This paper presents the design and first matched evaluation of the Advanced-Dungeon, the most demanding stage of a multi-level, narrative-driven workshop concept that guides learners from block-based to text-based and finally to authentic, project-scale programming within a shared open-source 2D dungeon environment. The transition between visual and textual programming is a well-documented barrier in early computer science education, and existing approaches address only the first step or treat constructionist game-making as a Scratch-only activity, which limits the reachable concepts. The Advanced-Dungeon engages pupils aged 15 to 17 in open-ended problem-solving directly within the game's source code. They read, debug, and extend a real Java codebase, implement algorithms such as pathfinding and sorting, and build Portal-inspired puzzles that require vector arithmetic and angle calculations. Two introductory formats, the Blockly-Dungeon and the Java-Dungeon, share the same game world and prepare learners for this final stage. Across all Advanced-Dungeon iterations, 36 learners completed the pre-survey and 29 the post-survey. Participants reported high enjoyment ($M=4.52$ on a 1 to 5 scale), rated the difficulty as appropriate, and wanted to attend further workshops. In open answers they highlight modifying real game code as the most rewarding part and name concrete gains in arrays, vectors, and abstract classes. An individually matched analysis ($n=10$ pairs) refines earlier results: where individual change can actually be measured, programming interest is stable rather than rising. All workshop materials and the dungeon environment are available as open educational resources and open-source software.

Keywords: Game-based learning, Narrative-centered learning, Programming workshops

1. Introduction

Early computer science education faces a persistent transition gap between visual block-based programming, introductory text-based coding, and more open, project-scale programming (Alrubaye et al., 2019; Weintrop, 2019). Block-based environments such as Scratch and Blockly lower the entry barrier and foster early computational thinking (Resnick et al., 2009; Fraser, 2015; Grover and Pea, 2013), yet learners who move beyond them often encounter a sharp discontinuity when they are first asked to read, write, and modify authentic source code. Existing bridging tools such as Droplet (Bau, 2015) or PyBlox (Jean et al., 2025) cover the first step toward textual syntax, but rarely extend into realistic, project-scale programming contexts.

In previous work the authors introduced a broader workshop concept with three progressively more demanding formats inside a shared 2D video game (Matutat et al., 2026). The present paper narrows the focus to the most advanced level, the Advanced-Dungeon, and its design as a Learning by making games environment in the constructionist tradition of Papert (1980) and Kafai and Burke (2015). Learners no longer solve isolated exercises; they open the game's real source code, locate bugs, implement missing behaviour, and observe the consequences in the running game world. This links constructionist game making (Kafai and Burke, 2015) with narrative game-based learning (Min et al., 2020).

The contribution is a design description of this advanced stage and a first matched evaluation of its effects.

2. Workshops

The workshop concept comprises three formats built on the same dungeon environment; full details are in Matutat et al. (2026). The Blockly-Dungeon introduces programming basics through a browser-based visual editor (Resnick et al., 2009; Weintrop, 2019). The Java-Dungeon drives the same levels through simplified Java code. Learners can switch between the Blockly view and its Java equivalent. Unlike purely generative hybrids (Alrubaye et al., 2019), the Java-Dungeon requires learners to write and edit text themselves, which supports the paradigm transition emphasised by Weintrop (2019). The shared level geometry and narrative across both formats reduce cognitive load once textual syntax appears and bridge into the Advanced-Dungeon.

3. Advanced Dungeon

The Advanced-Dungeon targets pupils aged 15 to 17 with prior experience in Java. In contrast to the two introductory formats, it is not a curated sequence of exercises. Learners open the actual source code of the dungeon environment in a suitable IDE and play through levels whose behaviour is fully defined by the code in front of them but contains bugs or lacks the implementation of specific functionality.

This positions the Advanced-Dungeon as a Learning by making games environment, in the literature usually described as constructionist gaming (Kafai, 1995; Kafai and Burke, 2015). The underlying assumption is that learners build knowledge most effectively when they create or modify artefacts that are personally meaningful (Harel and Papert, 1991). Most classroom implementations rely on Scratch or comparable block systems, which restricts the reachable concepts to small, self-contained projects. The Advanced-Dungeon transfers the principle into an authentic software engineering setting: a real, non-trivial Java codebase rather than a scripted sandbox, with inheritance, component composition, exceptions, and professional tooling. This aligns the format with research on game modding as a learning context, where players treat the game itself as the artefact to be modified (El-Nasr and Smith, 2006).

A typical scenario starts inside the game. Players encounter a broken mechanic, for example a door that will not open. Riddles and hints in the level point to the relevant code. Learners trace the call chain, formulate a hypothesis, and implement a fix, an activity that mirrors the fact-finding model of program comprehension (LaToza et al., 2007); hot reloading propagates the edit within seconds. More demanding tasks require new functionality: extending components, adding data structures, implementing algorithms such as pathfinding or sorting, building player controls, buttons, pressure plates, and portal logic, with applied mathematics such as vector arithmetic and angle calculations.

Figure 1 shows a concrete example. The player faces a green portal that cannot be used because its exit point is not computed. A hint points to 'MyCalculations', where 'calculatePortalExit' is missing. Learners add the implementation in Figure 2: identify the paired portal by name, read its position and view direction, and return the tile in front of it. On save, hot reloading makes the portal usable. The task combines utility classes, conditional logic, object references, and vector translation in a single method.

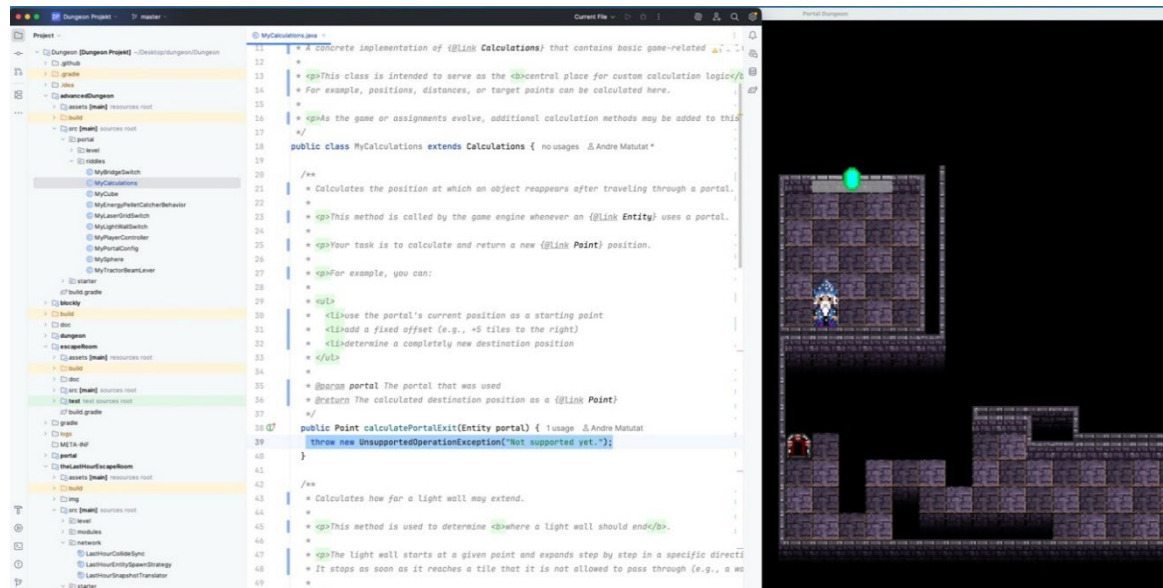


Figure 1: Advanced-Dungeon portal task. Left: the IDE with the open 'MyCalculations' class. Right: the game showing the player in front of an unusable green portal. Once 'calculatePortalExit' is implemented, the portal links to its blue counterpart at the bottom right and the level can be solved

```

public Point calculatePortalExit(Entity portal) {
    Entity otherPortal;

    if (portal.name().equals(PortalUtils.BLUE_PORTAL_NAME))
        otherPortal = Tools.getPortal(PortalUtils.GREEN_PORTAL_NAME);
    else otherPortal = Tools.getPortal(PortalUtils.BLUE_PORTAL_NAME);

    Point portalPosition = Tools.getPosition(otherPortal);
    Direction portalViewDirection = Tools.getViewDirection(otherPortal);
    return portalPosition.translate(portalViewDirection);
}

```

Figure 2: Learner solution for the portal exit task, added to `MyCalculations` and picked up by hot reloading

Learning is task-driven; instructors act as facilitators and encourage pair programming, consistent with evidence on collaborative novice programming (Hanks et al., 2011).

The didactic centre shifts from solving a given problem to shaping an existing system; progress is measured by observable game behaviour. The Advanced-Dungeon thus extends the Blockly-to-Java progression into a third stage in which the object of instruction is a maintained game project.

4. Evaluation and Limitations

Pooled across iterations, $n=36$ completed the pre-survey and $n=29$ the post-survey (retention 80.6%). Participants were predominantly aged 15 to 17, 28 male, 5 female, 3 non-binary.

Self-reported interest in programming on a 1 to 5 Likert scale rose in aggregate from $M=4.31$ ($SD=1.06$, $Mdn=5$) to $M=4.69$ ($SD=0.54$, $Mdn=5$); the drop in dispersion is consistent with differential dropout. An individually matched analysis on the subset with self-generated participant codes ($n=10$ pairs) gave pre $M=4.50$ ($SD=0.50$), post $M=4.40$ ($SD=0.66$), mean difference -0.10 ($SD=0.30$); an exact Wilcoxon signed-rank test is non-significant ($W=0$, $p=1.00$) but underpowered. Nine of ten pairs were identical, one decreased by one scale point. The aggregate increase therefore reflects sample composition rather than individual change, a limitation we now make explicit.

Post-only items used a 1 to 5 agreement scale (1 = do not agree at all, 5 = strongly agree): enjoyment $M=4.52$ ($SD=0.51$, $n=29$), understanding the concepts $M=4.38$ ($SD=0.73$, $n=29$), willingness to attend a further workshop $M=4.28$ ($SD=0.70$, $n=29$), expecting to engage more with programming $M=4.08$ ($SD=0.95$, $n=25$), appropriately challenging $M=4.00$ ($SD=0.98$, $n=22$). All means lie above the scale midpoint of 3.

Open answers most frequently name arrays, vectors, and abstract classes as gains; learners highlight "reading source code that someone else wrote" and "modifying the code of the game while playing it" as the most enjoyable part, matching the constructionist rationale. The main improvement request is more guidance for orienting in the unfamiliar codebase.

These results are design feedback rather than generalisable effects: voluntary participation invites self-selection (ceiling at $Mdn=5$), the matched $n=10$ is exploratory only, outcomes are self-reported with no skill assessment and no control group, and item-level n varies due to partial post-survey completion.

5. Conclusion

The Advanced-Dungeon shows that constructionist game-making can be lifted from Blockly-style sandboxes into a real Java codebase used by 15 to 17 year olds. Acceptance is high and learners report concrete gains in algorithms, data structures, and code navigation. The matched analysis tempers earlier reports of rising programming interest: where individual change can be measured, interest is stable. Next iterations will add scaffolding for codebase orientation and broaden the matched sample.

Acknowledgements

This project was carried out at Bielefeld University of Applied Sciences and funded by the European Union and the State of North Rhine-Westphalia as part of the EFRE/JTF funding program "Pakt für Informatik 2.0" (Funding Code: EFRE-20300105).

Ethics Declaration: The workshops were voluntary, non-curricular educational events. All surveys were anonymous and collected no personally identifying information. No sensitive, medical, or biometric data were gathered. Participation was based on informed consent by the pupils and, where required by age, their legal guardians, and could be withdrawn at any time.

AI Declaration: Generative AI tools were used to support language editing and improve clarity and conciseness of the manuscript; all content, interpretations, and conclusions remain the sole responsibility of the authors.

References

- Alrubaye, H., Ludi, S. and Mkaouer, M.W., 2019. Comparison of block-based and hybrid-based environments in transferring programming skills to text-based environments. arXiv:1906.03060.
- Bau, D., 2015. Droplet, a blocks-based editor for text code. *Journal of Computing Sciences in Colleges*, 30(6), pp.138-144.
- El-Nasr, M.S. and Smith, B.K., 2006. Learning through game modding. *Computers in Entertainment*, 4(1), Article 7.
- Fraser, N., 2015. Ten things we've learned from Blockly. In: 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). IEEE, pp.49-50.
- Grover, S. and Pea, R., 2013. Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), pp.38-43.
- Harel, I. and Papert, S. eds., 1991. *Constructionism*. Norwood, NJ: Ablex Publishing.
- Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L. and Zander, C., 2011. Pair programming in education: A literature review. *Computer Science Education*, 21(2), pp.135-173.
- Jean, D., Stein, G., Broll, B. and Ledeczi, A., 2025. Easing the Block-to-Text Transition: A Scaffolded Approach to Learning Python. In: *Proceedings of the ACM Global on Computing Education Conference 2025*, Vol. 1. New York: ACM, pp.113-119.
- Kafai, Y.B., 1995. *Minds in Play: Computer Game Design as a Context for Children's Learning*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Kafai, Y.B. and Burke, Q., 2015. Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist*, 50(4), pp.313-334.
- LaToza, T.D., Garlan, D., Herbsleb, J.D. and Myers, B.A., 2007. Program comprehension as fact finding. In: *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. New York: ACM, pp.361-370.
- Matutat, A., Franke, B., Schumann, N., George, B.C., Keuntje, J.-M. and Gips, C., 2026. Designing Workshops to Bridge Visual and Textual Programming: A Narrative Game Approach. In: *Proceedings of the 10th Annual International GamiFIN Conference 2026*. New York: ACM. Available at: <https://doi.org/10.1145/3789624.3789625> (Accessed: 24 April 2026).
- Min, W., Mott, B., Park, K., Taylor, S., Akram, B., Wiebe, E., Boyer, K.E. and Lester, J., 2020. Promoting Computer Science Learning with Block-Based Programming and Narrative-Centered Gameplay. In: 2020 IEEE Conference on Games (CoG). Piscataway, NJ: IEEE, pp.654-657.
- Papert, S., 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.
- Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y., 2009. Scratch: Programming for all. *Communications of the ACM*, 52(11), pp.60-67.
- Weintrop, D., 2019. Block-based programming in computer science education. *Communications of the ACM*, 62(8), pp.22-25.