From TikTok to Hard Fun

Progressive Engagement in Computational Thinking through Game Design

Alexander Repenning[1]

[1] PH FHNW, School of Education, Brugg-Windisch, Switzerland

Abstract

This paper examines how constructionist learning principles can be maintained in an era of shortened attention spans and passive content consumption. While TikTok's own research shows that 50 % of users find videos over 60 seconds stressful, we demonstrate that meaningful creative engagement remains possible through carefully designed learning progressions. Using data from STEAM Discovery Fairs, we analyze how students across different grade levels and genders engage with RULER.game, a Computational Thinking Tool enabling rapid creation of programmable video game characters. Our findings reveal that while overall engagement levels were high, boys and girls derived enjoyment from different aspects – girls primarily from design elements and boys from programming components. The study demonstrates that by providing appropriate scaffolding and tools, students can quickly transition from passive consumers to active creators, even within perceived modern attention constraints. These results suggest that constructionist principles remain viable when learning designs thoughtfully balance cognitive and affective challenges while accounting for gender-specific preferences in creative computing activities.

Keywords and Phrases: computer science education research, block-based programming, programming by example.

1. Introduction

The foundational principles of constructionism emphasize children as active creators of their learning experiences, particularly in mathematics and programming. However, today's reality shows an increasing trend toward passive, consumption-oriented technology use, even in educational settings. This shift away from creation is further amplified by the evolution of information technologies – from the web to search engines to generative AI – which increasingly think for us rather than serve as "objects to think with" as Seymour Papert (Papert, 1980) originally envisioned.

The history of constructionist tools like Logo highlights this tension. In 1999, Papert himself acknowledged the need for evolution (Kahn, 2007), stating "I could not agree more with you about current Logo being out of date and am planning to immerse

Repenning, A. (2025). From TikTok to Hard Fun. Progressive Engagement in Computational Thinking through Game Design. *Constructionism Conference Proceedings, 8/2025*, 227–238. https://doi.org/10.21240/constr/2025/48.X

myself in thinking about what a language for 2005 (or so) could be." While many modern programming environments exist, they often fail to fully embrace contemporary affordances that could support creative learning.

This challenge is particularly visible in the context of mobile devices. Allan Kay's vision (Kay, 1972) of the Dynabook – a portable, collaborative computing device for children – has technically been achieved, yet these devices face increasing resistance in educational settings. Many schools and districts are moving to ban smartphones, raising questions about whether the challenges lie in the technology itself or in how society employs these tools.

The concept of "hard fun" (Papert, 1985) offers a framework for understanding this challenge, suggesting the need for a balance between cognitive and affective challenges through rewarding activities that connect with students' interests. However, current trends in social media, particularly platforms like TikTok, threaten this balance. Research shows that nearly 50 % of TikTok users find videos longer than 60 seconds stressful (Stokel-Walker, 2022), with many watching content at double speed. This "TikTok generation" exemplifies a societal shift where sustained engagement with challenging tasks is increasingly replaced by rapid-fire, algorithm-driven content consumption.

To explore how modern tools could bridge the gap between consumption and creation, we investigated approaches that could effectively support students' transition from passive consumers to active producers with two research questions:

> *RQ1: How do engagement levels and understanding of programming differ across grade levels and between genders?*
>
> *RQ2: What are the gender-specific preferences and prior experiences that might influence future programming participation?*

Through STEAM (Science Technology Engineering Art and Math) Discovery Fairs, we tested how quickly students could transition from consumers to creators using the RULER.game Computational Thinking Tool. Students completed a creative cycle in just 15 minutes: drawing video game characters, digitizing them, and bringing them to life through basic programming. This compressed timeline demonstrated that meaningful engagement in computational thinking (Saqr et al., 2021) and creative production does not necessarily require lengthy introduction periods.

2. Related Work

Constructionist learning theories (Kafai et al., 1996; Papert et al., 1993) emphasize that effective learning occurs through building meaningful artifacts, requiring programming tools that provide key affordances to help learners overcome *cognitive challenges*. Creating Logo, one of Papert's goals was to make programming more accessible by mitigating syntactic programming challenges. Block-based programming introduced by AgentSheets (Repenning et al., 1996) and popularized with Scratch (Resnick et al., 2009) completely eliminated syntax errors. However, block-based programming environments like Scratch face several key challenges despite their benefits in removing syntactic barriers. While they effectively prevent syntax errors (Mouza et al., 2020) through their jigsaw puzzle-like interface (Glinert et al., 1984) research reveals that students still encounter significant logical and semantic

difficulties. Ben-Yaacov et al. (Ben-Yaacov et al., 2023) found that 39% of student program executions resulted in errors related to issues like counting and orientation misconceptions rather than syntax. The debugging process is particularly challenging as many block-based environments provide limited debugging support (Hromkovic et al., 2021) and may intentionally suppress error messages (Deiner et al., 2023). Additionally, block-based programming tools have been criticized for potentially leading to code smells and inefficient programming patterns, as demonstrated in studies of Scratch programs (Hermans et al., 2016). These challenges highlight that while block-based programming eliminates syntactic barriers, semantic and logical programming challenges likely outweigh syntactic ones.

Equally important to cognitive challenges are *affective challenges* (Picard et al., 2004) in computational learning environments. Constructionist learning emphasizes building personally meaningful artifacts, but the nature of these artifacts and how tools and learning designs can scaffold their creation requires careful consideration. Instead of abstract computation like calculating prime numbers, programming can enable creative expression through diverse mediums: controlling robots, telling interactive stories through environments like Alice (Kelleher et al., 2007), building scientific simulations with NetLogo (Wilensky et al., 2015), programming robots (Ajaykumar et al., 2021), designing games (Kafai, 1995; Overmars, 2004; Spieler et al., 2018), creating interactive E-textiles (Spieler et al., 2020), and programming toys (Janka, 2008). If the goal is to achieve Papert's vision of "hard fun," it becomes essential to scaffold what Dewey (Dewey, 1913) called "instrumental motivation" – where learners see clear connections between their efforts and meaningful outcomes.

However, even with strong initial motivation, providing low-threshold tools like block-based programming is insufficient if the tools and learning designs don't facilitate a "gentle slope" progression from initial engagement to more sophisticated creation. Alan Perlis captured this challenge in his warning about the Turing tarpit (Perlis, 1982): "Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy." The journey from TikTok-style passive consumption to meaningful "hard fun" creation requires carefully designed learning progressions that balance both cognitive and affective challenges at each step. This progression must maintain engagement while gradually increasing complexity, allowing learners to build confidence and competence simultaneously.

3. Approach: Accessibility-to-Challenge Progression

To conceptualize a progression towards hard fun it is necessary to have a sense of space outlining the interaction between cognitive and affective challenges. This framework provides a lens through which we can analyze both grade-level progression in programming understanding and gender-specific engagement patterns, directly addressing our research questions about how different groups of students transition from consumers to creators. Figure 1 describes what we call the *Accessibility-to-Challenge Progression* (4a – 4b).

The framework builds on our *Cognitive/Affective Challenges* space (Repenning, 2016), which emerged from research exploring challenges preventing learners from transitioning from "have to" to "want to" programming. This model is particularly relevant as we attempt to engage today's students who, according to research by TikTok itself, find videos longer than 60 seconds stressful and often watch content at double speed (Stokel-Walker, 2022). These shortened attention spans make it even

more critical to understand how programming activities can be structured to overcome both cognitive barriers (making programming accessible) and affective barriers (making programming exciting) simultaneously.

A key insight from exploring the Cognitive/Affective Challenges space is that motivation and cognition in game design (Walter et al., 2007) are deeply interconnected. Even if students are initially motivated to create a game or simulation, if the tools are too complex, they'll likely abandon the effort because the return on investment isn't clear. Conversely, if programming is made too simplistic without meaningful creation possibilities, students may complete tasks but won't develop the "hard fun" engagement that Papert described. The Accessibility-to-Challenge Progression represents a pathway for thoughtfully balancing these dimensions as students develop programming competency

The Cognitive/Affective Challenges space can be illustrated through four characteristic examples: (1) Writing C++ programs to compute prime numbers represents the classic "hard and boring" scenario – it poses significant cognitive challenges for novice programmers while offering minimal intrinsic motivation for most people except the already mathematically inclined. (2) Cleaning up a room exemplifies low cognitive challenge but high tedium – its Sisyphean nature, where effort is quickly undone, makes it difficult to maintain intrinsic motivation. (3) Doom scrolling through TikTok represents the "easy and exciting" extreme – demonstrated by millions of users spending hours consuming short-form content, though such passive entertainment typically offers limited developmental value. (4) The Accessibility-to-Challenge Progression is well illustrated by crossword puzzles, particularly the New York Times series that gradually increases in difficulty from Monday to Saturday, showing how cognitive challenge can increase while maintaining engagement through carefully calibrated progression.

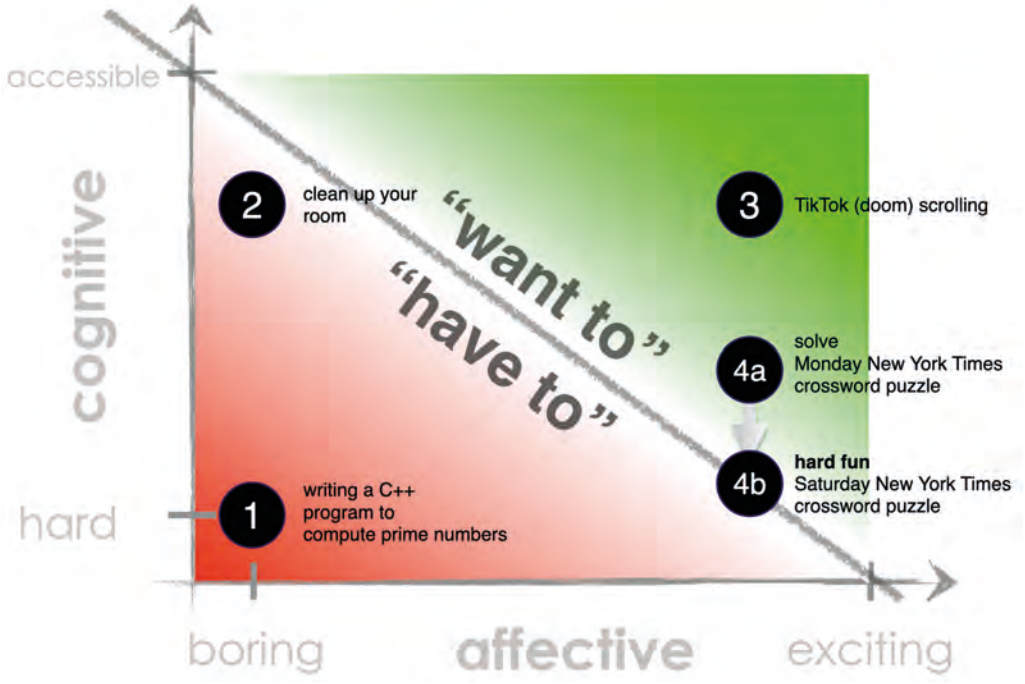

Figure 1: Developing a Accessibility-to-Challenge Progression towards "hard fun".

While these examples illustrate the challenge space, they also suggest a pathway forward. The critical need is to help students transition from the passive consumption habits of TikTok (3) to engaging in meaningful creative challenges (4) through carefully designed affordances that go beyond mere syntactic support.

4. Tool: RULER.game

RULER.game (Figure 4, left) was developed as a *Collaborative Computational Thinking Tool* to facilitate the transition from consumers to producers by enabling students to create their own video games through what we call *proxy-based programming* (Repenning et al., 2024). This approach affords *pragmatic prebugging* – proactive debugging support that helps prevent logical errors before they occur. Comparing to established error rates of block-based programming (Ben-Yaacov et al., 2023), our study shows that pragmatic prebugging results in error rates ten times lower than traditional block-based programming (Repenning et al., 2024).

Following a long line of Computational Thinking tools (Repenning, 2017) including AgentSheets as the first block-based programming tool (Repenning et al., 1996), AgentCubes as a 3D Computational Thinking Tool adding pragmatic debugging support to block-based programming (Repenning, 2013a), RULER.game introduces proxy-based programming to make programming more "accessible and exciting" (Repenning, 2013b).

Recent research on programming languages supporting children's programming has focused heavily on syntax while largely neglecting pragmatics. Pragmatic debugging programming support – approaches that scaffold understanding "what code means in specific situations" – can be found in foundational work exploring programming by example (Cypher, 1993; Lieberman, 2001; Repenning et al., 2001), and live programming (McDirmid, 2007, 2013). In programming by example (Figure 2, bottom) a human changes a situation, e.g., by interacting with objects in a game, to have the computer change the code. Conversely, in live programming (Figure 2, top) the human changes code to have the computer update the situation. Without a proper mechanism to define a focus both programming paradigms can be difficult to control by the user (Ferdowsifard et al., 2020). Proxy-based programming, in contrast, employs the notion of a proxy object serving as Logo turtle-like embodiment through body syntonicity (Watt, 1998) helping users to focus and to understand causal connections between code and situation. Combined, the human and the computer use this proxy in a sandbox to proactively explore future situations (Figure 2, middle). Imagine a student programming a game of Pac-Man. That student would select Pac-Man in its current situation and start programming. The Pac-Man would be cloned into a visible proxy object looking like the a ghosted Pac-Man image. The student can now issue various actions to see possible future situations by observing the proxy execute these actions. Unlike with basic testing functions found in programing tools like Logo, AgentSheets, Scratch or AgentCubes, this experimentation takes place in a protected sandbox preventing unwanted side effect or the need to tediously snap code pieces apart for the sole purpose of testing.

Proactive proxy-based programming will reveal programming errors the instance they are made. That is, users do not have to write and then execute a complete programs such as programs to solve a Hour of Code like challenge (Repenning et al.,

2016) or the Kodetu challenges (Ben-Yaacov et al., 2023). Users can fix the error and RULER.game will replay the proxy actions to get ready for more corrected instructions.

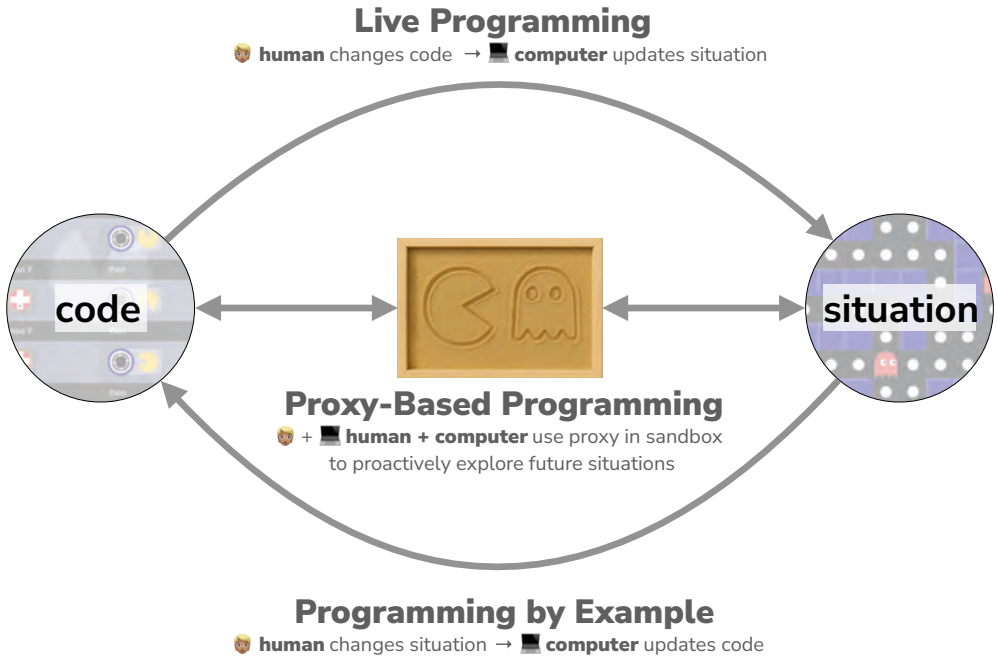

Figure 2: Proxy-Based Programming uses a embodied proxy in a sandbox to explore possible code and situation futures.

RULER.game helps overcome the Turing tar-pit (Perlis, 1982) through the integration of AI actions that make sophisticated behaviors accessible. With the RULER.game AI actions even 1st graders (Repenning, 2024) were able to use sophisticated AI pathfinding (Repenning, 2006) to program ghosts to collaboratively pursue Pac-Man through a maze. In other programming languages employed to make games, and aimed at kids, such as Scratch, this level of game AI would be essentially impossible, forcing programmers to resort to less compelling game mechanics such as ghosts moving toward Pac-Man while ignoring walls, or implementing random or scripted movement patterns.

5. Intervention: Create Your Own Video Game

To explore the research questions we evaluated a compact learning design for STEAM Discovery Fairs having kids create the start of a simple video games in about 15-20 minutes:

1. 1st Table: draw your own video game character on paper (Figure 3, left)
2. 2nd Table: create RULER.game project on iPad (Figure 3, right)
3. scan in game background. Some background images were provided (Figure 3, right)
4. scan in own video game character
5. learn how to program simple rules to make character controllable through directional gamepad

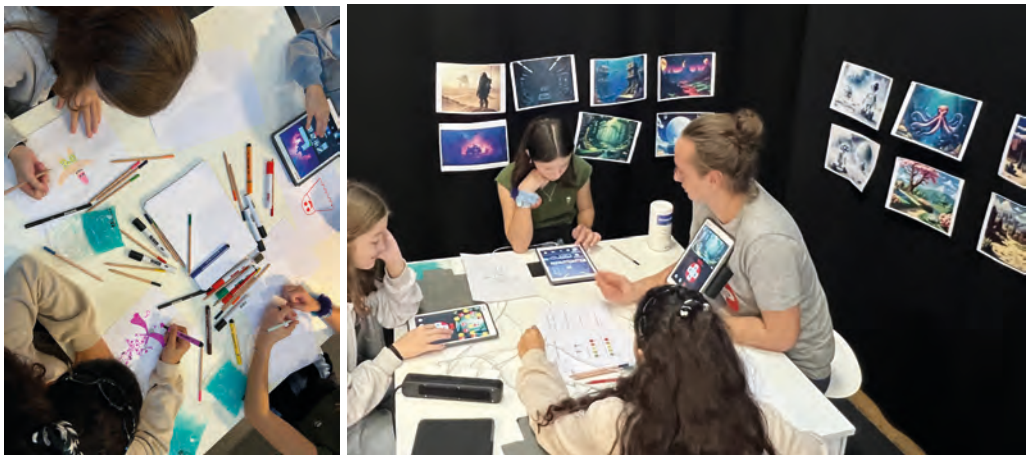

Figure 3: Left: 1st Table: Draw your own character. Right: 2nd Table. Programming and play testing

It was not clear if a station focusing on programming would be sufficiently attractive to kids especially when surrounded by a vast array of activities constantly competing with their attention. The tunSolothurn.ch 2024 event offered 46 hands-on experiments across 24 research stations, allowing over 8,000 children and youth to engage with STEM activities in creative and engaging ways. The diverse offerings included creating handmade cooling pads, programming robots, experimenting with virtual reality, assembling electronic pins, constructing bridges from popsicle sticks, building metal puzzles, and participating in "explosive" science shows.

The station, aided by 2-3 pre-service teacher students from the School of Education, FHNW Switzerland, was barely able to keep up with the barrage of kids eager to make their own video games. In the few moments where there was no line waiting to participate, kids were allowed to continue with programming to create additional characters, and to extend programming, e.g., with collision detection or score counting. The minimal expectation was to have kids create a game with one character that they could move around the world (Figure 4).

Figure 4: Left: RULER.game game with one character programmed. Right: Example characters drawn.

Many kids returned to the station trying to continue with programming. Some came back waiting for a multi-person opening to work on a game with their friends. Some kids returned on different days with teachers, parents or even grandparents.

6. Evaluation

To evaluate the learning design we collected the games produced and asked students to fill out a short questionnaire. The "Create Your Own Game" station operated for 3 days during which over 300 games were created. To provide some opportunity for students to continue with game design at home, or school, projects were uploaded to the RULER.game server. Students received a RULER.game flyer with information for parents and teachers including a printed QR code label providing a link to their current project.

Before leaving the station N = 189 students filled out a short questionnaire with the following questions:

1. What grade are you in? (number)
2. What is your gender? (male/female)
3. Enjoyment: I enjoyed creating a game: strongly disagree: -2, disagree: -1, agree: 1, strongly agree: 2
4. Understanding: I understand how to program: strongly disagree: -2, disagree: -1, agree: 1, strongly agree: 2
5. Retention: I would like to continue programming: yes/no
6. where would you like to continue to program
7. What kinds of games are you currently playing? (open)
8. What part of the activity did you like best? (open)
9. What part of the activity did you find difficult? (open)
10. Have you programmed before? yes/no
11. What did you program before? (open)

Exploring *RQ1: "How do engagement levels and understanding of programming differ across grade levels and between genders?"* We found a mild scissor effect (Figure 5) where engagement decreases while understanding increases (after grade 2). The crossover with enjoyment and the very high self-reported levels of understanding could be outliers due to the small sample size of students from grade 1 (5 girls and 3 boys).

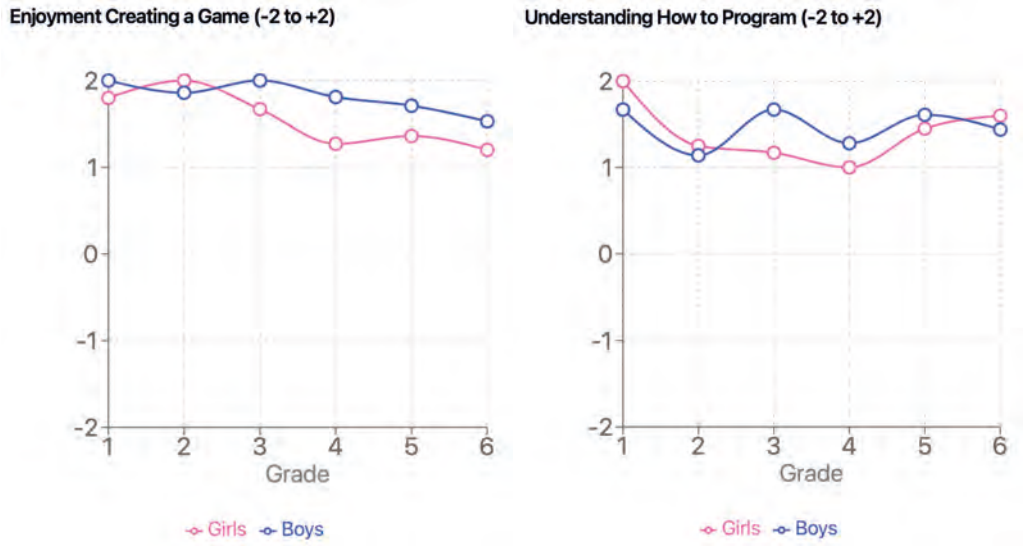

Figure 5: Scissor effect: enjoyment (Q3) decreases (left), while understanding (Q4) increases (right).

Exploring *RQ2: "What are the gender-specific preferences and prior experiences that might influence future programming participation?"* We looked at retention, that is, the claimed interest to continue with programming in Figure 6 left. This retention suggested by the number of students wanting to continue to program is high but even larger for boys (88.9% n = 135) compared to girls (67.4% n = 46). 8 students did not identify gender. Interestingly, the percentage of girls in 6th grade reporting prior programming experience (60%, Figure 6, right) is almost 3 times larger than the percentage of the boys with prior programming experience (23%).

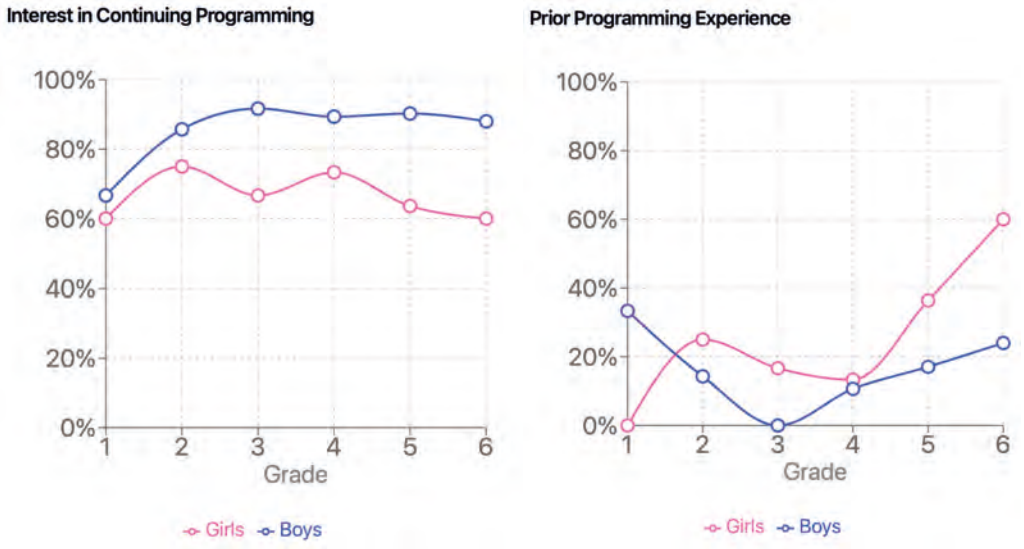

Figure 6: Interest (Q5) in continuing to program (left) versus prior programming (Q10) experience (right).

Looking at more depth into gender specific preferences we found substantial differences in the most enjoyed aspects of the activity. As an open question students mention a big array of reasons including "everything" but there were two coded clusters called "design" and "programming."

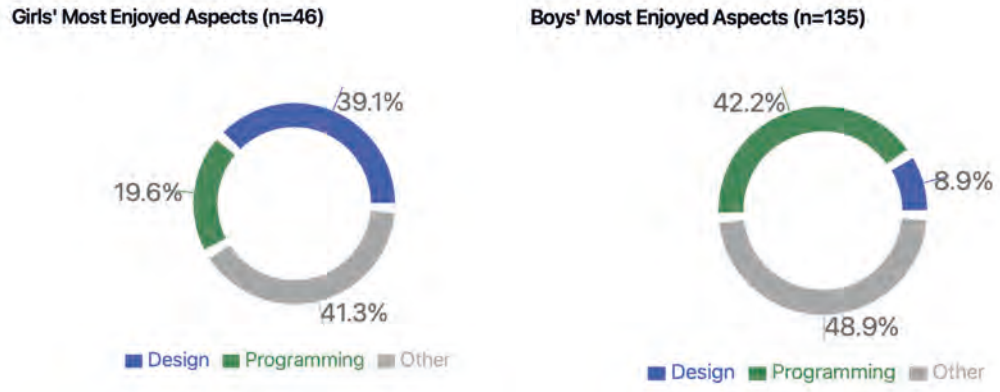

Figure 7: (Q8) Girls enjoy design most (left), buys enjoy programming most (right).

Figure 7 shows a flip with two highly significant gender differences (design: $p < 0.001$, programming: $p = 0.006$) and very large effect sizes (Hill et al., 2008) between the two categories. Both "design" with an effect size (Cohen's d) of 3.97 and "programming" with an effect size of -3.14 demonstrate these pronounced differences. In summary, while game design activities are enjoyed by both genders comparably, their sources of enjoyment differ substantially.

Limitations to the evaluation included the somewhat informal nature of the intervention and that there was no control group and the length of intervention varied widely. Additionally, as a science fair setting, some students self-selected to participate in this activity among many competing options, potentially biasing our sample toward those already interested in programming or game design. The informal learning environment also made it difficult to control for prior knowledge and experience, though we attempted to account for this through our questionnaire. The high-energy atmosphere of the science fair, with 46 different hands-on experiments competing for attention, meant that some students may have rushed through the activity. While our pre-service teacher facilitators were trained on the activity, their varying levels of experience with the tool could have influenced outcomes. Finally, our reliance on self-reported data through the questionnaire, particularly for measures like understanding and enjoyment, may not fully capture actual learning outcomes.

7. Conclusions

Our findings challenge assumptions about the incompatibility between TikTok-era attention spans and constructionist learning. Through our STEAM Discovery Fair intervention, even first-grade students successfully created their own video games using the RULER.game Computational Thinking Tool in about 15 minutes. While the activity's initial focus on drawing video game characters made it broadly approachable, our analysis revealed important engagement patterns across grade levels and genders. Overall enjoyment was high, but boys and girls derived satisfaction from different aspects – girls primarily valuing design elements and boys favoring programming components. The tool's proxy-based programming approach made programming accessible while maintaining constructionist principles of personal creation. High retention rates and students returning for additional sessions suggest that with appropriate scaffolding and tools, active creation can successfully compete with passive consumption habits. These results demonstrate that constructionist learning principles remain viable when learning designs thoughtfully balance cognitive and affective challenges while accounting for diverse creative preferences.

References

Ajaykumar, G., Steele, M., & Huang, C.-M. (2021). A survey on end-user robot programming. *ACM Computing Surveys (CSUR), 54*(8), 1-36.

Ben-Yaacov, A., & Hershkovitz, A. (2023). Types of Errors in Block Programming: Driven by Learner, Learning Environment. *Journal of Educational Computing Research, 61*(1), 178-207.

Cypher, A. (1993). *Watch What I Do: Programming by Demonstration*. Cambridge, MA: MIT Press.

Deiner, A., & Fraser, G. (2023). NuzzleBug: Debugging block-based programs in scratch. *arXiv preprint arXiv:2309.14465*.

Dewey, J. (1913). *Interest and effort in education*: Houghton Mifflin.

Ferdowsifard, K., Ordookhanians, A., Peleg, H., Lerner, S., & Polikarpova, N. (2020). *Small-step live programming by example.* Paper presented at the Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology.

Glinert, E. P., & Tanimoto, S. L. (1984). Pict: An Interactive Graphical Programming Environment. *IEEE Computer*, 265-283.

Hermans, F., & Aivaloglou, E. (2016). *Do code smells hamper novice programming? A controlled experiment on Scratch programs.* Paper presented at the 2016 IEEE 24th International Conference on Program Comprehension (ICPC).

Hill, C. J., Bloom, H. S., Black, A. R., & Lipsey, M. W. (2008). Empirical benchmarks for interpreting effect sizes in research. *Child development perspectives, 2*(3), 172-177.

Hromkovic, J., & Staub, J. (2021). The Problem with Debugging in Current Block-based Programming Environments. *Bulletin of EATCS, 135*(3).

Janka, P. (2008). *Using a programmable toy at preschool age: Why and how*. Paper presented at the Teaching with robotics: didactic approaches and experiences. Workshop of International Conference on Simulation, Modeling and Programming Autonomous Robots.

Kafai, Y. B. (1995). *Minds in play: Computer game design as a context for children's learning*. Hillsdale, N.J.: L. Erlbaum Associates.

Kafai, Y. B., & Resnick, M. (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. Mahwah, NJ: Lawrence Erlbaum Associates.

Kahn, K. (2007). Should LOGO keep going forward 1? *Informatics in Education-An International Journal, 6*(2), 307-321.

Kay, A. C. (1972). *A personal computer for children of all ages*. Paper presented at the Proceedings of the ACM National Conference.

Kelleher, C., Pausch, R., Pausch, R., & Kiesler, S. (2007). *Storytelling alice motivates middle school girls to learn computer programming*. Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems.

Lieberman, H. (2001). *Your Wish Is My Command: Programming by Example*. San Francisco, CA: Morgan Kaufmann Publishers.

McDirmid, S. (2007). *Living it up with a live programming language*. Paper presented at the OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, Montreal, Quebec, Canada.

McDirmid, S. (2013). *Usable Live Programming*. Paper presented at the SPLASH Onward!, Indianapolis, Indiana.

Mouza, C., Pan, Y.-C., Yang, H., & Pollock, L. (2020). A multiyear investigation of student computational thinking concepts, practices, and perspectives in an after-school computing program. *Journal of Educational Computing Research, 58*(5), 1029-1056.

Overmars, M. (2004). Teaching computer science through game design. *Computer, 37*(4), 81- 83.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.

Papert, S. (1985). Hard Fun. Retrieved from http://www.papert.org/articles/HardFun.html

Papert, S., & Harel, I. (Eds.). (1993). *Constructionism*. Norwood, NJ: Ablex Publishing Corporation.

Perlis, A. J. (1982). Special feature: Epigrams on programming. *ACM SIGPLAN Notices, 17*(9), 7-13.

Picard, R. W., S., P., Bender, W., Blumberg, B., Breazeal, C., Cavallo, D., . . . Strohecker, C. (2004). Affective Learning – a Manifesto. *BT Technology Journal, 22*(4), 253-269.

Repenning, A. (2006). *Collaborative Diffusion: Programming Antiobjects*. Paper presented at the OOPSLA 2006, ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon.

Repenning, A. (2013a). *Conversational Programming: Exploring Interactive Program Analysis*. Paper presented at the 2013 ACM International Symposium on New ideas, New Paradigms, and Reflections on Programming & Software (SPLASH/Onward! 13), Indianapolis, Indiana, USA.

Repenning, A. (2013b). Making Programming Accessible and Exciting. *IEEE Computer, 18*(13), 78-81.

Repenning, A. (2016). *Transforming "Hard and Boring" into "Accessible and Exciting"*. Paper presented at the NordiCHI 2016, Workshop on Cultures of Participation in the Digital Age: From "Have to" to "Want to" Participate, Gothenburg, Sweden.

Repenning, A. (2017). Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. *Journal of Visual Languages and Sentient Systems, 3*(July), 68-89.

Repenning, A. (2024). *Escaping the Turing Tar-Pit with AI Programming Blocks*. Paper presented at the The 19th WiPSCE Conference on Primary and Secondary Computing Education Research, Munich, Germany.

Repenning, A., & Ambach, J. (1996). *Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing*. Paper presented at the 1996 IEEE Symposium of Visual Languages, Boulder, CO.

Repenning, A., & Basawapatna, A. (2016). *Drops and Kinks: Modeling the Retention of Flow for Hour of Code Style Tutorials*. Paper presented at the The 11th Workshop in Primary and Secondary Computing Education (WiPSCE 2016), Münster, Germany.

Repenning, A., & Basawapatna, A. (2024). *RULER: Prebugging with Proxy-Based Programming*. Paper presented at the IEEE Symposium on Visual Languages and Human-Centric Computing, Liverpool, UK.

Repenning, A., & Perrone-Smith, C. (2001). Programming by Analogous Examples. In H. Lieberman (Ed.), *Your Wish Is My Command: Programming by Example* (Vol. 43, pp. 351-369): Morgan Kaufmann Publishers.

Resnick, M., Maloney, J., Monroy-Hernndez, A., Rusk, N., Eastmond, E., Brennan, K., . . . Kafai, Y. (2009). Scratch: programming for all. *Commun. ACM, 52*(11), 60-67. doi:10.1145/1592761.1592779

Saqr, M., Ng, K., Oyelere, S. S., & Tedre, M. (2021). People, Ideas, Milestones: A Scientometric Study of Computational Thinking. *ACM Transactions on Computing Education (TOCE), 21*(3), 1-17.

Spieler, B., Krnjic, V., Slany, W., Horneck, K., & Neudorfer, U. (2020). *Design, code, stitch, wear, and show it! mobile visual pattern design in school contexts.* Paper presented at the 2020 IEEE Frontiers in Education Conference (FIE).

Spieler, B., & Slany, W. (2018). Game development-based learning experience: Gender differences in game design. *arXiv preprint arXiv:1805.04457*.

Stokel-Walker, C. (2022). TikTok Wants Longer Videos – Whether You Like It or Not. *Wired*.

Walter, S., Barron, B., Forssell, K., & Martin, C. (2007). *Continuing Motivation for Game Design*. Paper presented at the CHI 2007, San Jose, California, USA.

Watt, S. (1998). *Syntonicity and the Psychology of Programming.* Paper presented at the Proceedings of the Tenth Annual Meeting of the Psychology of Programming Interest Group, Milton Keenes, UK.

Wilensky, U., & Rand, W. (2015). *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*: MIT Press.