Talk2Play: Voice-Based Game Logic Authoring for Children

ANONYMOUS AUTHOR(S)

We present Talk2Play, a playful voice-based game creation system that helps children design game worlds, define game objects, and author interactive game logic through natural language. Instead of requiring children to begin with formal programming syntax, Talk2Play lets students describe characters, objects, rules, controls, and behaviors through speech while the system translates those descriptions into structured game logic.

Talk2Play emphasizes game design and computational thinking concepts such as entities, properties, events, conditions, actions, loops, and player controls. Children can define objects such as players, structures, enemies, items, and backgrounds; assign properties and behaviors to them; and express simple rules such as what should happen when a button is pressed, when objects collide, or when a condition becomes true. The system also supports AI-assisted asset creation so that students can generate or customize visual game elements while keeping the main authoring focus on game logic and interaction design. To avoid building a new block-programming engine from scratch, Talk2Play currently translates natural-language game rules into MakeCode Arcade-compatible block code.

We describe the current system design, interaction workflow, implementation status, and planned evaluation with children.

CCS Concepts: • **Human-centered computing** → **User interface design**; • **Applied computing** → **Interactive learning environments**; • **Software and its engineering** → *Visual languages.*

Additional Key Words and Phrases: children, game creation, voice-based authoring, game logic, constructionism, creative play, AI-assisted authoring, visual programming, game-based learning

Fig. 1. Intended sample workflow for Talk2Play. A child draws or selects game objects on a tablet and describes their game logic through natural language, such as defining a Goomba as an enemy entity. The system uses this input as the starting point for generating structured game logic and editable block-code behavior.

2026. Manuscript submitted to ACM

1 Introduction

Children often engage deeply with games and stories, but the process of creating interactive games can require programming concepts that are difficult to access at early stages. Talk2Play explores a voice-based authoring model in which children can design games by speaking about objects, rules, controls, and interactions.

The goal of Talk2Play is to make game creation feel closer to conversation, storytelling, and play. Children can create or select characters and objects, describe what those objects are, and define behaviors using voice commands. For example, a student might say that an object is a player, that a block is a structure, or that pressing a button should make a character jump. In this way, the system introduces game programming ideas such as entities, properties, conditions, loops, controls, and events through interaction rather than through syntax-first instruction.

This work-in-progress paper presents the early design and implementation of Talk2Play. We focus on the system concept, interaction workflow, current prototype status, and planned evaluation.

Fig. 2. Example natural-language entity definition in Talk2Play. A child identifies Kirby as an entity of type player, allowing the system to map the drawn or selected sprite to a semantic game role before later attaching controls, properties, and interaction rules.

2 Related Work

Talk2Play builds on constructionist learning, block-based programming, game creation systems, and AI-assisted multimodal authoring. Constructionist learning emphasizes that children develop powerful ideas by creating personally meaningful artifacts [11]. Systems such as Scratch and ScratchJr lower the barrier to programming through visual blocks [2, 15], while Alice and Storytelling Alice use 3D animated worlds and narrative authoring to introduce programming through creative production [1, 7]. These systems show the value of child-authored computational media, but students still generally need to manually select, arrange, and debug programming constructs.

Game-centered learning environments introduce programming through interactive play. Kodu Game Lab supports character behavior through condition-action rules [4, 18], while AgentSheets, AgentCubes, and Scalable Game Design teach computational thinking patterns such as collision, generation, transport, and agent behavior through game and simulation creation [6, 13, 14]. Talk2Play follows this view of game design as a pathway into programming, but shifts the first step from manually choosing constructs to describing intended behavior in natural language. The generated logic is then exposed as MakeCode Arcade-compatible block code for inspection, revision, and extension [8, 9].

Talk2Play is most directly inspired by DrawTalking, which explores building interactive worlds through sketching and speaking [16]. Our system follows this multimodal direction, but focuses specifically on educational game design and explicit programming concepts. Students define typed game entities such as players, enemies, items, projectiles, structures, and backgrounds, then attach properties, controls, conditions, loops, collisions, and behaviors to them. For example, a child might say that ‘Goomba is an enemy” and that it should ‘move left and right forever,” or that “when

Kirby touches the burger, health increases by five." These utterances are translated into structured game logic and editable block code.

Our work also relates to game-creation platforms such as GDevelop and Flowlab, which provide accessible tools for building games without traditional programming syntax [3, 5]. These platforms support event-based or visual game logic, but still assume that learners can express rules inside the platform's logic system. Talk2Play instead uses natural language as a bridge from everyday descriptions into computational representations. Speech is not a replacement for programming, but a starting point for generating logic students can later refine through blocks.

Recent AI-assisted authoring systems also inform our design. Meta's Animated Drawings system shows how children's hand-drawn characters can be segmented, rigged, and animated [17]. Capybara extends this direction into AI-enabled augmented reality, allowing children to create, customize, and program 3D characters using text-to-3D generation, auto-rigging, body tracking, and vision-based interaction with physical objects [19]. Talk2Play shares this emphasis on children as creators rather than passive consumers of AI-generated content, but focuses on 2D sprite-based platformer games where programming concepts are more explicit and inspectable.

This game-logic focus creates a secondary learning opportunity. As children describe movement, collision, scoring, health changes, achievements, unlocks, and progression, they practice giving precise instructions to computational systems. These mechanics resemble familiar patterns from sandbox and progression-based games such as Minecraft and Terraria, where players encounter authored worlds, resources, achievements, and gradual unlocking of abilities or areas [10, 12].

Overall, Talk2Play extends prior work by combining children's drawings and generated sprites as personal game objects, natural-language descriptions as the entry point for game logic, and block-code transcription as an inspectable program representation. Rather than treating AI as an automatic game generator, the system uses AI and NLP as scaffolding for a child-centered authoring process across speech, drawing, blocks, and play.

3 Methodology: NLP-to-Block-Code Pipeline

Talk2Play uses a staged pipeline that converts children's spoken or typed game descriptions into editable block-style game logic. Figure 3 shows the current engineering flow. The prototype begins with a child command entered through the web interface, either as speech or typed text. In the intended speech-based interaction, the audio is first transcribed into text, then passed to an NLP layer that extracts the game intent.

The NLP stage maps each utterance into a constrained game-logic schema. This schema identifies entities, semantic types, actions, events, properties, and conditions. For example, a rule such as "when Kirby touches the burger, increase health by five" is represented as a collision event between two entities, followed by a property update. Similarly, a command such as "Goomba is an enemy and moves left and right forever" is represented as an enemy entity with a horizontal movement behavior.

After intent extraction, the system stores the result as a structured intermediate representation. This representation acts as a bridge between open-ended natural language and executable game logic. Rather than generating arbitrary code directly, the prototype compiles this representation into MakeCode Arcade-compatible TypeScript. This allows the generated logic to be inspected as block-style code, revised by the student, and eventually tested in a game simulator.

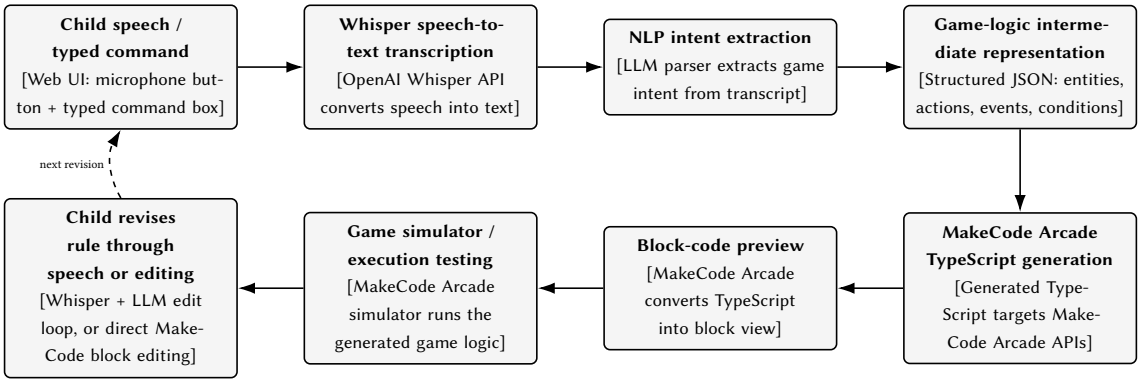

Fig. 3. Engineering pipeline for Talk2Play. Child speech or typed commands are transcribed and parsed into game intents, converted into a structured game-logic intermediate representation, and compiled into MakeCode Arcade-compatible TypeScript. The generated code is shown as block-style logic for review, with simulator execution and revision loops planned for full end-to-end testing.

After intent extraction, the system stores the result as a structured intermediate representation. This representation acts as a bridge between open-ended natural language and executable game logic. Rather than generating arbitrary code directly, the prototype compiles this representation into MakeCode Arcade-compatible TypeScript. This allows the generated logic to be inspected as block-style code, revised by the student, and eventually tested in a game simulator.

At this stage, the NLP-to-block-code transcription pipeline is functional for a controlled set of entity definitions, type assignments, movement behaviors, collision events, and property changes. Full game execution, simulator handoff, runtime debugging, and systematic playtesting are still in progress.

After intent extraction, the system stores the result as a structured intermediate representation. This representation acts as a bridge between open-ended natural language and executable game logic. Rather than generating arbitrary code directly, the prototype compiles this representation into MakeCode Arcade-compatible TypeScript. This allows the generated logic to be inspected as block-style code, revised by the student, and eventually tested in a game simulator.

Figure 4 shows an example movement rule. A child describes an enemy behavior in natural language, and the system translates the command into block-code logic that creates the Goomba as an enemy sprite, sets its horizontal velocity, and enables wall-bounce behavior so that it moves left and right across the map.

Figure 5 shows a collision-and-property rule. In this example, the child defines a conditional interaction in which Kirby touching a burger increases health. The generated blocks create the player and item sprites, initialize the health property, and add an overlap event that updates health and removes the collected item.

4 Prototype Interface and Interaction Flow

Figure 6 shows the main Talk2Play editing interface. The top transcription bar displays the current natural-language rule, in this case defining behavior for the cloud object. The left toolbar provides common editing tools such as play, copy, delete, undo, redo, camera toggle, and save. The center canvas is the main authoring area, where students arrange objects and select sprites before defining behaviors through the microphone control. The right panel lists the available scene objects and their semantic types. This object-type view is important for introducing object-oriented programming concepts, since students can see that game elements are not only visual sprites, but typed entities such as players, enemies, structures, and backgrounds. In this example, Mario is selected as an object and highlighted on the canvas.

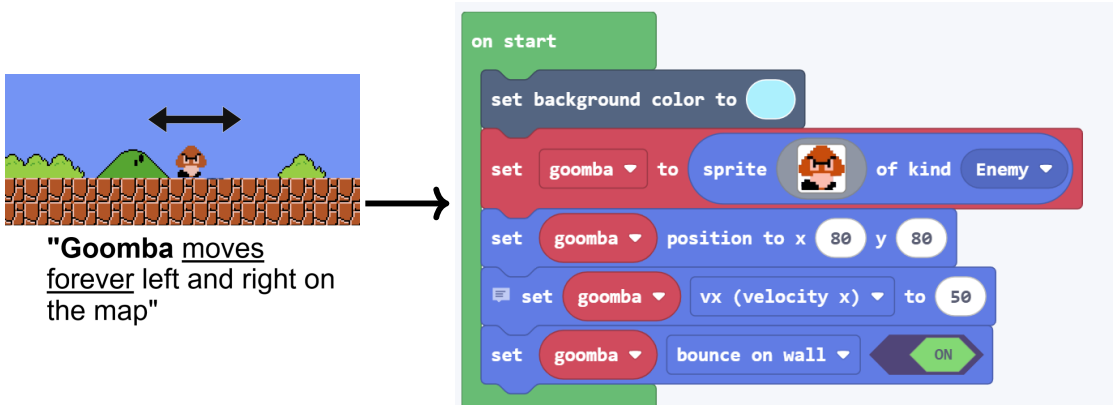

Fig. 4. Example movement-rule transcription. A student describes that Goomba moves left and right forever on the map, and Talk2Play converts the natural-language rule into MakeCode Arcade block logic for enemy creation, horizontal velocity, and wall-bounce behavior.

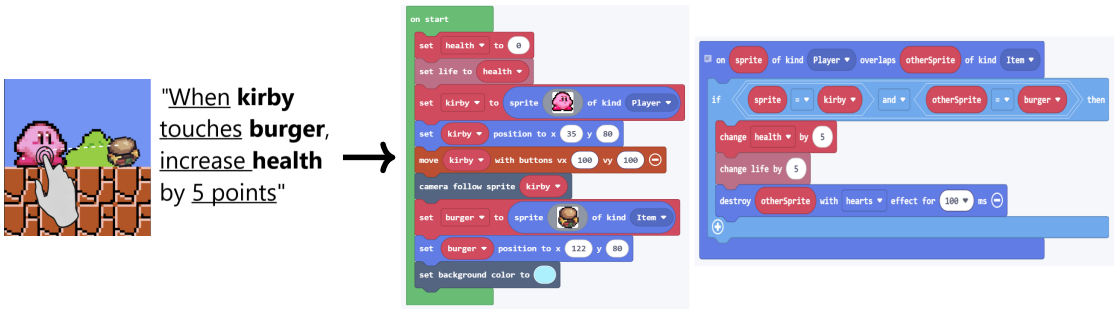

Fig. 5. Example collision-and-property transcription. A student describes that Kirby should gain health when touching the burger, and Talk2Play converts the rule into MakeCode Arcade blocks that define the sprites, initialize health, detect the overlap event, update the property, and remove the item.

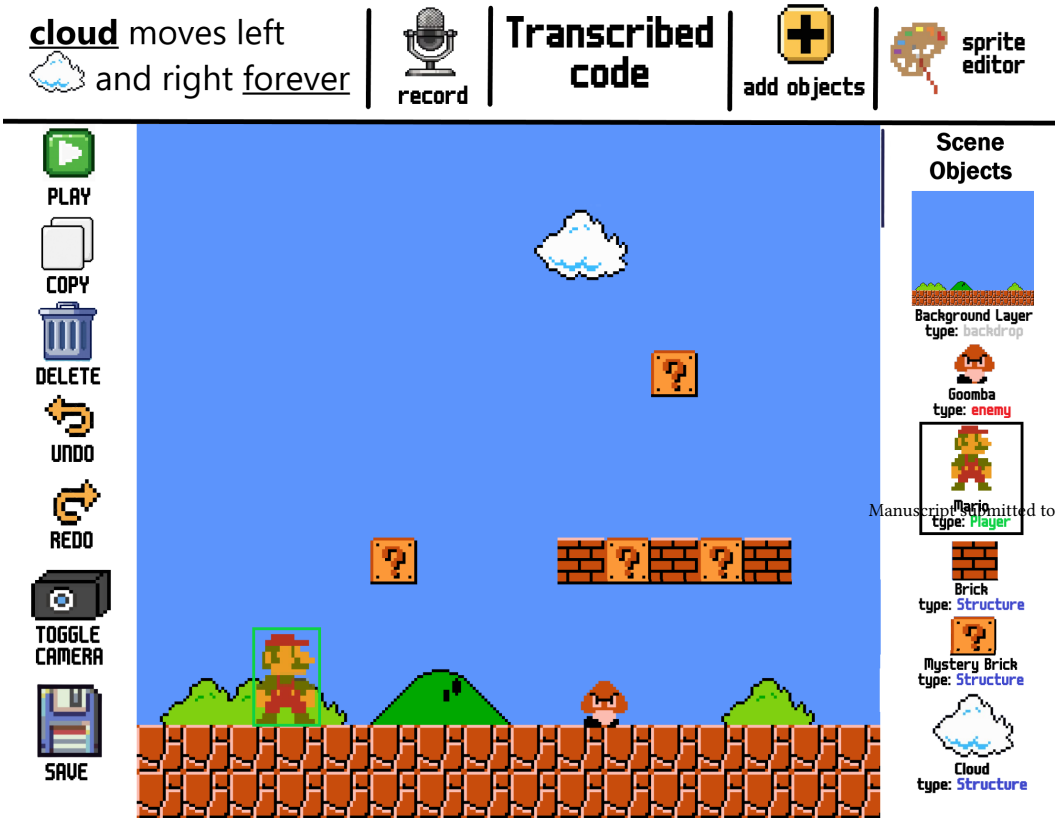

Figure 7 shows the add-object interface. When students select the add-objects button, the canvas is tinted and the available object palette appears over the scene. Objects are grouped by semantic role, including players, structures, enemies, and items. This organization helps students connect visual game assets to programming concepts: a sprite is not only an image, but an entity with a type that determines how it can behave in the game. Students can add new objects to the scene, then further define their behavior through speech, typed commands, or later block-code editing.

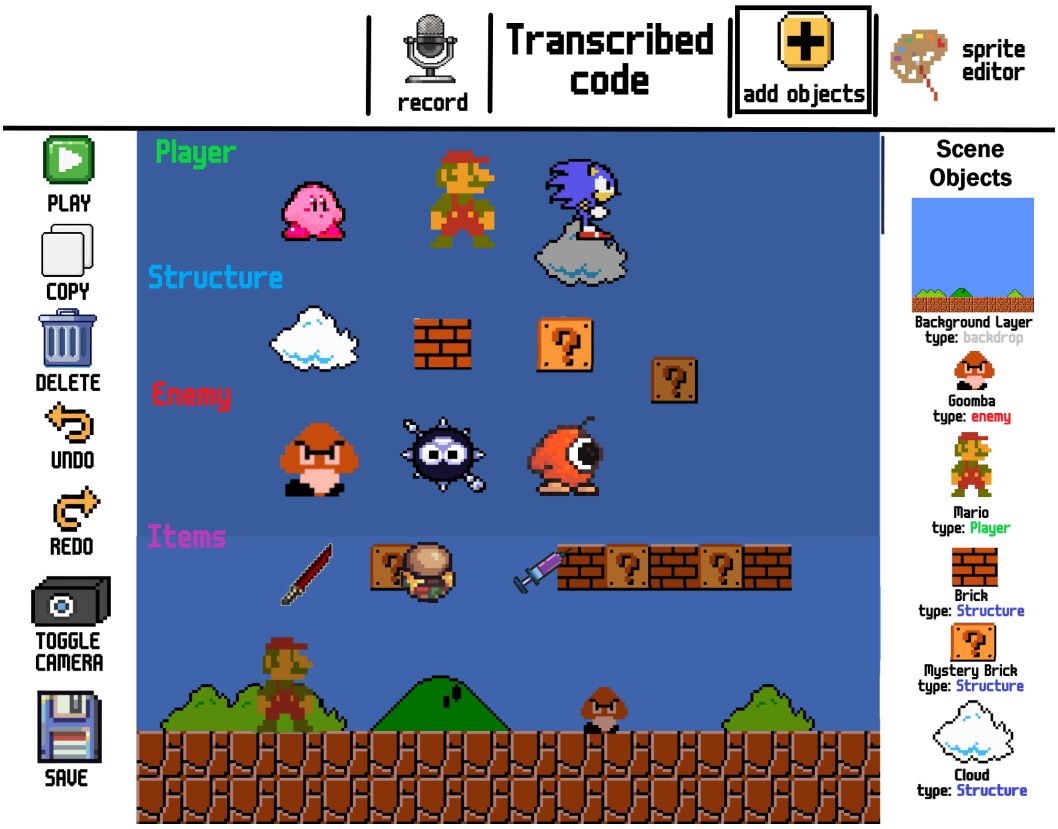

Fig. 7. Add-object interface. The object palette groups available sprites by semantic type, allowing students to add players, structures, enemies, and items while maintaining a visible connection between game assets and object-oriented game concepts.

Figure 8 shows the prototype in Play Mode. Once the play button is pressed, the editing tools collapse into a simplified playable game view. Any executable rules recorded in the transcribed code logic are activated during this mode. The A and B buttons serve as programmable action controls, such as jump, attack, interact, collect, or use an item, depending on the student's spoken rules. The directional pad provides movement controls; if movement or collision rules have been defined, they are evaluated during play, including platform contact, wall boundaries, enemy collisions, or hazard triggers. The Stop button exits Play Mode and returns to the expanded editing interface, while Restart restores the game scene to the initial state defined by the user.

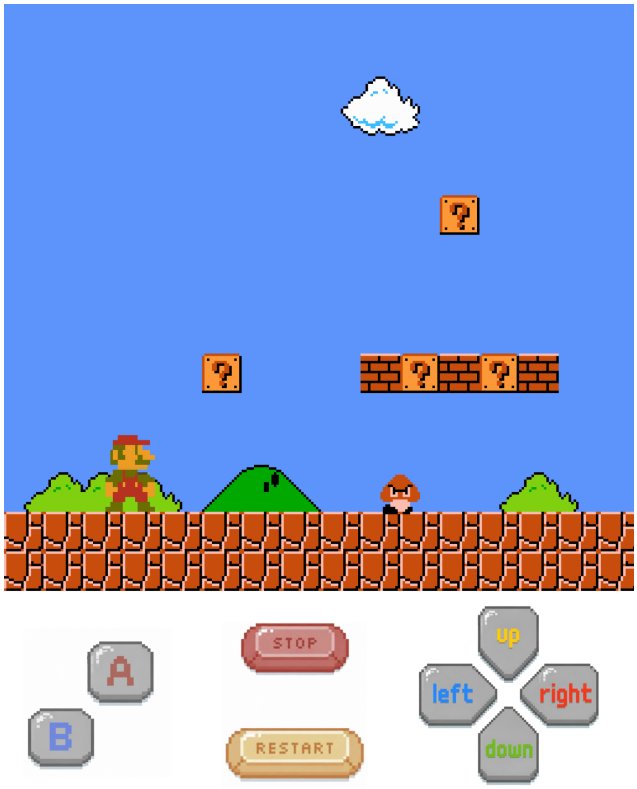

Fig. 8. Play Mode interface. The editing tools collapse into a simplified game-testing view with programmable A/B buttons, directional movement controls, Stop for returning to editing mode, and Restart for resetting the user-defined scene.

5 Conclusion

Talk2Play is a work-in-progress system for helping children create 2D platformer-style games through voice-based authoring and AI-assisted design. The prototype currently supports natural-language descriptions of objects, controls, and rules, translating them into structured game logic and MakeCode Arcade-compatible blocks.

The current system remains limited in several ways. Its voice prompts and logic library are primarily designed for simple 2D platform games, and more complex conditions will require a larger voice-to-code library and more carefully designed rule structures. Character movement is also mostly static, unlike mature 2D platform games, the prototype does not yet support full animated movement sequences, power-ups, projectile animations, or richer player–NPC interactions. Future work will explore AI-assisted generation of coherent sprite/frame sequences from a student's baseline drawing, and how students might define physics, so game artwork feels consistent rather than random.

References

[1] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, Kevin Christiansen, Robert Deline, Jim Durbin, Rich Gossweiler, Shuichi Koga, Chris Long, Brian Mallory, Steve Miale, Kevin Monkaitis, Jeffrey Patten, James Pierce, Joseph Shochet, David Staack, Brian Stearns, Richard Stoakley, Chris Sturgill, John Viega, Jeff White, George Williams, and Randy Pausch. 2000. Alice: Lessons Learned from Building a 3D System for Novices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. Association for Computing Machinery, New York, NY, USA, 486–493. doi:10.1145/332040.332481

[2] Louise P. Flannery, Elizabeth R. Kazakoff, Paula Bonta, Brian Silverman, Marina Umaschi Bers, and Mitchel Resnick. 2013. Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*. Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/2485760.2485785

[3] Flowlab. 2026. Flowlab Game Creator. [https://flowlab.io/

[4] Allan Fowler, Teale Fristoe, and Matthew MacLaurin. 2012. Kodu Game Lab: A Programming Environment. *The Computer Games Journal* 1, 1 (2012), 17–28. doi:10.1007/BF03392325

[5] GDevelop. 2026. GDevelop: Free, Fast, Easy Game Engine. [https://gdevelop.io/

[6] Andri Ioannidou, Alexander Repenning, and David C. Webb. 2009. AgentCubes: Incremental 3D End-User Development. *Journal of Visual Languages and Computing* 20, 4 (2009), 236–251. doi:10.1016/j.jvlc.2009.04.001

[7] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. 2007. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. Association for Computing Machinery, New York, NY, USA, 1455–1464. doi:10.1145/1240624.1240844

[8] Microsoft MakeCode. 2026. MakeCode Languages: Blocks, Static TypeScript and Static Python. [https://makecode.com/language

[9] Microsoft MakeCode Arcade. 2026. Blocks Language. [https://arcade.makecode.com/blocks

[10] Mojang Studios. 2026. Minecraft Official Site. [https://www.minecraft.net/

[11] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, NY, USA.

[12] Re-Logic. 2026. Terraria. [https://terraria.org/

[13] Alexander Repenning. 1993. Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. Association for Computing Machinery, New York, NY, USA, 142–143. doi:10.1145/169059.169119

[14] Alexander Repenning, David C. Webb, Kyu Han Koh, Hilarie Nickerson, Susan B. Miller, Catharine Brand, Ian Her Many Horses, Ashok Basawapatna, Fred Gluck, Ryan Grover, Kris Gutierrez, and Nadia Repenning. 2015. Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools Through Game Design and Simulation Creation. *ACM Transactions on Computing Education* 15, 2, Article 11 (2015), 31 pages. doi:10.1145/2700517

[15] Mitchel Resnick, John Maloney, Andres Monroy-Hernandez, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (2009), 60–67. doi:10.1145/1592761.1592779

[16] Karl Toby Rosenberg, Rubaiat Habib Kazi, Li-Yi Wei, Haijun Xia, and Ken Perlin. 2024. DrawTalking: Building Interactive Worlds by Sketching and Speaking. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3654777.3676404

[17] Harrison Jesse Smith, Qingyuan Zheng, Yifei Li, Somya Jain, and Jessica K. Hodgins. 2023. A Method for Animating Children's Drawings of the Human Figure. *arXiv preprint arXiv:2303.12741* (2023). arXiv:2303.12741 doi:10.48550/arXiv.2303.12741

[18] Kathryn T. Stolee and Teale Fristoe. 2011. Expressing Computer Science Concepts Through Kodu Game Lab. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 99–104. doi:10.1145/1953163.1953197

[19] Lei Zhang, Shuyao Zhou, Amna Liaqat, Tinney Mak, Brian Berengard, Emily Qian, and Andres Monroy-Hernandez. 2025. Capybara: Empowering Children to Create AI-Enabled Augmented Reality Experiences. *arXiv preprint arXiv:2508.08467* (2025). arXiv:2508.08467 doi:10.48550/arXiv.2508.08467