Strawbies: Designing an inexpensive, portable, and fluid tangible programming game

Author 1 Author 2 Author 3

ABSTRACT
Strawbies is a tangible programming game designed for children ages 5 and up. Players arrange physical programming tiles to guide an onscreen character (Awbie) through an infinitely expanding world displayed on an iPad screen. Strawbies is portable, inexpensive, and fluidly interactive—characteristics made possible with the use of an Osmo game system that includes a mirror to reflect images in front of the iPad through the front-facing camera. Here we describe a set of principles that guided our design process along with evaluation sessions with children in schools and a museum. The combination of inexpensive tangible tiles and real-time computer vision through a tablet computer opens new possibilities for tangible computer programming to reach a broader audience.

Categories and Subject Descriptors
H.5.m [**Information interfaces and presentation (e.g., HCI)**]: Miscellaneous.

Keywords
Children; programming; tangibles; games; strawberries.

1. INTRODUCTION
Strawbies is a tangible programming game designed for children ages 5 and up that combines the use of inexpensive tangible tiles and real-time computer vision through a stationary iPad tablet (Figure 1). Players arrange wooden tiles to guide an onscreen character (Awbie) through a rich and infinitely expanding world filled with obstacles, thieving rodents, and strawberries. Strawbies is portable, inexpensive, and fluidly interactive. These features are made possible by the use of an Osmo game system (playosmo.com) that uses a mirror to reflect images in front of the iPad through the front-facing camera. In creating Strawbies, we sought to explore new possibilities opened by this system to create an engaging and collaborative game for a wide range of ages and playstyles.

To this point, tangible programming systems have had to make tradeoffs between inexpensive materials, portability, and real-time interaction. Many tangible programming systems have opted to use "smart" tangible components with embedded electronics and power supplies (e.g. [8, 14, 28]). This has the advantage of offering real-time interactive systems that can be used anywhere, with the downside that these systems are relatively expensive to create and manufacture. Other tangible programming systems have instead opted to use computer vision and "passive" tangible blocks or tiles (e.g. [10]). But, the ability to continuously track and respond to programming blocks has either required a stationary overhead camera fixture or an interactive surface with built-in camera hardware. It is possible to instead use a mobile device like an iPad, but this requires a point-and-shoot style of interaction (e.g. [11, 12]) in which real-time interaction is sacrificed for the sake of portability. The Osmo's use of a mirror over the top of the iPad's front-facing camera makes it possible to obtain the best of both worlds by combining fluid, real-time interaction with a mobile device and low-cost tangible materials.

We believe that these characteristics open opportunities for designers to create engaging tangible programming systems that are accessible to a much broader audience. Our aim with Strawbies is to explore this design space. Early on we discovered that despite the innovation of the Osmo, achieving our design goals was more difficult than we anticipated. We struggled with both the game design and the programming block design over many iterations before we felt that we had achieved a system that was inviting, engaging, and easy for children to learn with. Testing sessions with children in homes, schools, out-of-school programs, and a children's museum offered us quick feedback and helped us articulate design principles to support a versatile and engaging programming game. In this paper we describe our design process in more detail and share insights from our evaluation sessions.

Figure 1. Strawbies app with tangible tiles, iPad, and Osmo game system (from playosmo.com).

2. RELATED WORK

Our project builds on a long and rapidly growing tradition of programming environments for young children [3, 4, 7, 15, 18, 20, 21, 23, 22, 24, , 27, 28] (see also [14] for an excellent account of older work dating back to the 1970s). There is also growing momentum around the idea of supporting computational literacy activities throughout K-12 education, starting at the earliest grade levels. Our project contributes to this space by blending the flexibility, portability, and practicality of tangible programming with an open-ended formatting, which enables our game to be highly responsive to the spontaneity in early childhood learning.

3. DESIGN PRINCIPLES

Strawbies began as an exploration into using the Osmo to create a tangible programming environment. As part of our design process we settled on several principles that articulated both a set of pedagogical values and the kind of experience we hoped to create.

Fun and Inviting: We sought to create a game that was entertaining, attractive, and not intimidating. We hoped that the tangible objects would attract kids' attention from a distance and draw them into an engaging experience.

Collaborative: Research in the learning sciences suggests that kids spontaneously form diverse and effective patterns of collaboration in game play [16, 26]. Our observations of children playing Strawbies corroborated these findings with siblings, classmates, and friends engaging in a surprising variety of joint activity. It was important, therefore, that our game create space for diverse collaborative arrangements with two, three, and even four or more.

Responsive: It was important for the game to support fluid and real-time interaction. As much as possible we wanted the game to "responds immediately, clearly and predictably to users' actions." [25]

Simple but Complex: Even though we limited ourselves to a relatively small library of programming tiles, we wanted to create enough complexity in the combination of these blocks for the game to remain compelling for longer play sessions or more experienced players.

Developmentally Aligned: We wanted to ensure that the content of our game was aligned with the cognitive, perceptual-motor, and social ability of our target audience.

Portable: The system must be easy to setup and transport to classrooms, museums, homes, after-school programs, summer camps, etc.

Learning Goals

Our target age range is 5 and up. Our learning goals were structured for children who have had little or no experience with programming languages.

Sequencing: The game should familiarize and reinforce players with the concept of using sequences of blocks to construct plans.

Intrinsic Integration: Intrinsic Integration [6] is a principle from the learning sciences that argues that the best and most fun parts of a game should also be where the important learning is taking place. This is in contrast to using incentives to motivate players to engage in activities that would otherwise be seen as tedious.

Slow Learning Progression: We didn't want to force kids to create longer, more complex programs right away. In fact, it's possible to play the game using only one or two blocks at a time, never picking up a repeat block and never creating a sequence. Instead, we sought to subtly encourage (but never force) kids to explore new concepts, at their own pace, and only when they are ready.

4. DESIGN OVERVIEW

Our current iteration of Strawbies features an infinitely expanding world filled with obstacles, adversarial characters, and strawberries (Figure 1). Players slide and connect coding tiles in front of an iPad to guide Awbie on his quest for more strawberries. Obstacles include trees, cactuses, and water that can derail you from a strategically planned sequence of instructions. Large rodent characters will also steal berries from Awbie. Getting multiple strawberries with one sequence triggers Frenzy Mode, causing Awbie to go into a fun frenzy and eat even more strawberries. Lakes with lilypad paths and treasure islands are designed to encourage the use more complex sequences of programming tiles. Large bouncing strawberries are scattered throughout the world, which lead to short tutorials that show possible patterns using different sequences of blocks.

The game includes six action blocks: Up, Down, Right, Left, Rainbow, Flashlight. The direction blocks moved Awbie in the specified direction, Rainbow transported Awbie into a different part of the world, and Flashlight scared away rats. There were parameter blocks (1, 2, 3 or 4) that attach to action blocks and multiply the times an attached direction block executes. Repeat blocks can also be used to repeat the entire sequences of tiles multiple times depending on the quantifier block used. (Figure 2)

5. PARTICIPANTS

We formally evaluated our design with 29 children in two locations. Our first round of testing was with five kindergartners and five 2nd graders at a local elementary school (4 boys and 10 girls). Children participated in groups of two or three peers from the same grade, with each session lasting around 15 minutes. We started each session with minimal instruction and then allowed children to explore on their own. At the very end of each session, we prompted students to try repeat tiles and parameter values if they had not already.

Our second round of testing involved 7 sessions with 8 boys and 7 girls at a local children's museum (ages 5 to 10). Children in these sessions occasionally worked alone, but more often in groups of 2 or 3. There was no set time limit, but sessions generally lasted between 3 and 15 minutes, with one pair of siblings spending over 40 minutes (average session length 14:30). Parents were also invited to participate but, for the most part, allowed children to explore on their own. All sessions at both locations were video recorded with informed consent of parents and children.

We also conducted many rounds of informal testing with children at a local elementary school, two out-of-school programs, and in homes. We did not collect research data in these informal sessions but instead used them to test and refine our design.

6. DESIGN PROCESS AND EVALUATION

Our design process included several major iterations in which we explored different approaches for the iPad app and tangible

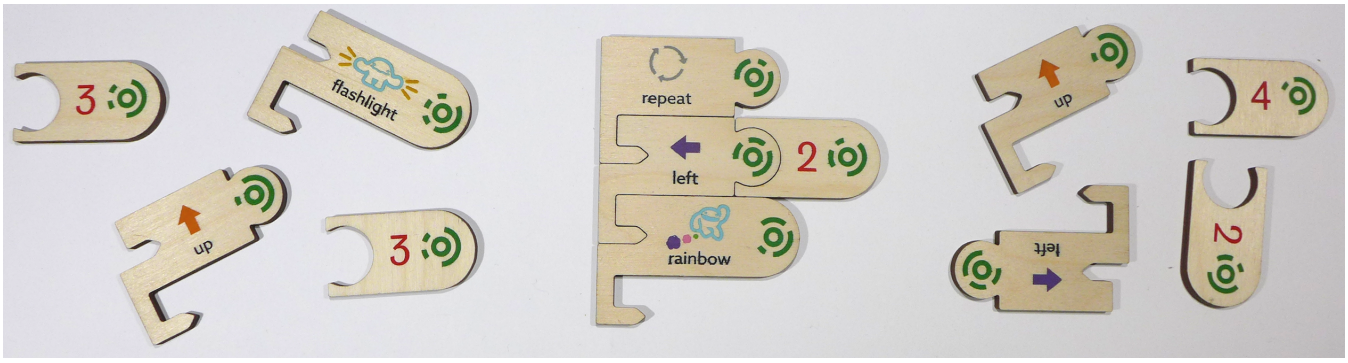

Figure 2. Collection of programming tiles available to guide the character in search of stawberries.

programming tiles. Here we describe specific categories of the design and important decisions.

Each feature generally began with ideas heavily borrowed from existing successful languages such as Scratch. As we tested these features, we modified and redesigned them as necessary.

Block Design

The most common actions when playing with a tangible programming environment is connecting and detaching coding blocks. We wanted to create blocks that could easily attach and detach, yet maintain structural integrity. We also wanted our blocks to intuitively connect together, without the need of any tutorials.

We went through several iterations of connection techniques, ultimately settling on two different connection types. The first connection technique is an arrow-hook that attaches action blocks together. This connector allowed action blocks to be attached on a surface only by sliding. The connection was also secure enough such that users could move attached blocks around on a surface without them coming apart. The design was intuitive for our users, as there is an arrow-shaped gap on each action block, invoking the action of connection. We experimented with two previous iterations of connection techniques: our first iteration was too cumbersome to attach, and our second iteration was too easily detached (Figure 3).

The second connection technique was inspired by the circular nature of the TopCode design. We used this connector to attach quantifier and action blocks. Our first iteration of blocks featuring TopCodes allocated a large amount of block space to the TopCode itself (Figure 3). By incorporating TopCodes into our connection technique, we were able to add illustrations to the blocks, without increasing the size of the blocks which was undesirable due to the already limited play space defined by the Osmo.

Evaluation

We observed two girls in Kindergarten playing Strawbies in a classroom for a 15 minute session. In this session, one girl, Player 1, had played Strawbies before, months back. The other girl, Player 2, had never played Strawbies. When the game started, Player 1 assumed full control of the game, immediately acquiring strawberries. This went on for almost 8 minutes, while Player 1 explained to Player 2 how to play the game. We observed that Player 1 was only playing the game using individual commands. While Player 2 played with blocks on the side, she started to construct more elaborate sequences, without prompts from the game. After 8 minutes, Player 2 suggested that they try out her constructed sequence.

Navigation

Navigating the world of Strawbies was one of the biggest roadblock for our players, and we went through several iterations to try and fix the problem (Figure 4). There were two important choices that we made: relative vs. absolute directionality, and separating vs. combining movement and direction blocks.

Separating out the action of turning one's body from the action of moving forward is appropriate for many programming systems (such as the Logo turtle). However, for the purposes of our game, we found that absolute directionality (face left, right, up, down) was easier for players to learn. Grammatically, keeping turning and walking separate also did not make sense. In English, people say “walk left”, but in our game, “Walk, Left” wouldn’t make

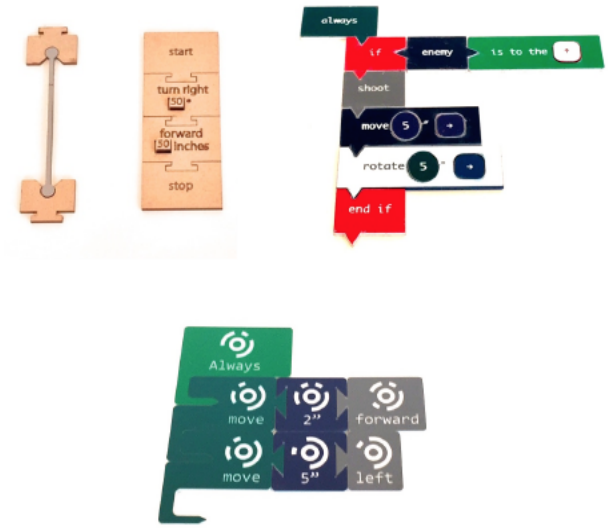

Figure 3. Our first three major iterations of block design.

sense; the character would first move in the direction that it is currently facing, then turn left. Lastly, combining movement and direction blocks is concise, reducing the number of blocks needed by half.

Evaluation

We observed two siblings, a boy and girl, playing Strawbies in a children's museum for 40 minutes. The siblings were collaborative, and often discussed where they should go with phrases like "oh look there's two strawberries down there!" and "no we should go left first then down this way we can get more". Their constructed sequences consisted of 1-3 blocks, often detaching and attaching different sequences to fit Awbie's situation. As the two siblings watched Awbie navigate their constructed sequences, we noticed that they were often mouthing each commands as Awbie executed them. This was something that became a common trend after we combined movement and direction blocks. The new commands are more concise and allow for more elaborate planning. We saw an increase in the complexity of constructed sequences upon combining movement and direction blocks.

Feedback

Children are constantly constructing conceptual models to explain how things in the world work [17]. We needed Strawbies to provide immediate, consistent, and clear feedback to our players, without which they would develop false beliefs about how the system worked. We tried to provide feedback about the blocks currently in the camera's field of view in a way that was immediate without being distracting or disruptive. Aside from the natural feedback provided by feeling and connecting tangibles, our feedback was limited to the digital space. Our passive tangibles, which are laser cut from wood, cannot provide any kind of 'smart' feedback.

We explored several options before settling on our current iteration. We first used half the screen to show a representation of what the camera sees (Figure 6). When a player slid a sequence in front of the iPad, a digital replica of the blocks would appear on the screen. After testing, we decided that this was redundant and took up too much screen space.

We then placed a live video-feed from the game's camera in the upper right corner, which circled TopCodes in red as they were recognized (Figure 6). We wanted this to bridge the connection between the physical and digital. Despite numerous verbal prompts during testing, we were unable to get children to pay attention to the video feed. Since it did not appear to help children understand what was going on, we removed this feature.

Our current iteration integrates feedback with gameplay. We project a path of Awbie's future actions onto the world (Figure 1, 6). The path animates incrementally from Awbie to visualize his potential movement. We felt that this was clear and immediate, but not disruptive or distracting. We hoped this projection will assist our players in the construction of more elaborate plans.

Evaluation

After the introduction of a projected path, children began modifying their sequence upon seeing different projections than what they had in mind. We also saw players construct more elaborate plans. We observed three Kindergarten students, 2 girls and 1 boy, in a classroom playing Strawbies for 15 minutes. 4

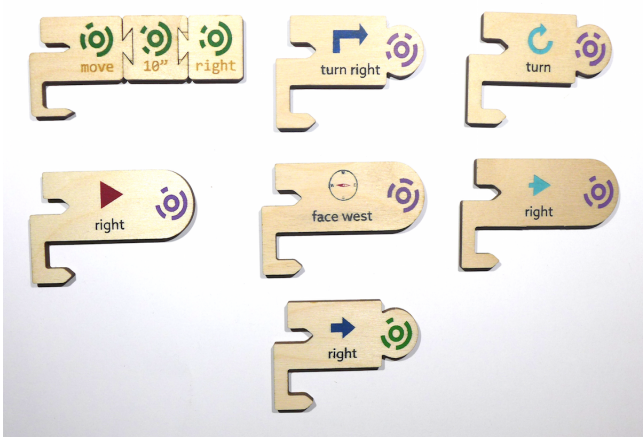

Figure 4. We quickly went through different symbols to find out which made the most sense to our players.

minutes into the game, the children were unable to compile their code, because they were all reaching for the screen and tapping the screen, blocking the TopCodes from being read. One girl shooed away the other players hands, and said "the path needs to be there!" (referencing the screen). 8 minutes into the game, the boy had a sequence that would have run Awbie into a rodent. The two girls yelled, "no! we don't want to touch it." The girls then replaced a direction block.

We are working on making our feedback more responsive. The path projection reacts quickly to blocks that are recognized (under 1.5 seconds), but it is not immediate. Without immediate feedback, some children will quickly move blocks around, hoping for something to happen on screen. This action however only exacerbates issues with the computer vision, causing feedback to no longer be clear or predictable.

Compiling

Many tangible programming environments provide users with input events that trigger the system to compile and run their code. The alternative is to have system continually evaluate code so that changes in arrangment of programming blocks is instantaneously reflected in a runtime environment. Using real-time computer vision opened both of these possibilities.

Our initial reaction was to keep as much interaction to the tangible interface as possible and evalute programs in real time. When blocks slid in front of the game's field of vision, the on-screen character immediately responded by following and continually looping those instructions. When blocks were moved away, the character immediately stopped.

The responsiveness of this real-time interactivity was immediate, but not clear or predictable. Our play testers had trouble maintaining control over their character, and often relied on using hands to cover up blocks when they wanted their character to stop; moving blocks outside of the camera's range was too tedious. The field of vision of the camera was also unclear. Sometimes, the camera would pick up blocks that a player forgot about, resulting in unexpected effects.

Our current iteration instead uses a tap on the screen as the triggering event (Figure 5). Players can easily reuse their sequence simply by re-tapping the screen. Once the screen is tapped, blocks can be removed from the game's vision without affecting the game.

The disadvantage of this mechanism is that when users tap the screen, their arm can block a portion of the image that is sent to the game. We tried to resolve this issue by making the tap box of the iPad almost the entire size of the screen, and added a one second delay to the game's image recognition.

Evaluation

We observed players to compile their code in different ways. Some tapped in the direction they wanted Awbie to go, and others tried to drag Awbie around the screen. We weren't worried about this behavior, as tapping to compile was an open enough mechanic such that many similar motions achieved the same effect. Eventually, as players constructed more elaborate sequences, tapping to compile became the predominant action.

Camera occlusion problems were more problemmatic. We observed two girls in Kindergarten playing Strawbies in a 30 minute session. The girls had no problem touching the screen to compile their code. However, for the first 10 minutes, they kept their finger on the screen in the direction of movement until the character stopped. It seemed that they thought Awbie was controllable by dragging as well. While one girl kept her finger on the screen, the other girl shifted blocks around below the girl's arm. The camera was not able to see the full picture because the blocks were occluded by the girl's arm, resulting in inconsistent and confusing actions from Awbie.

These kinds of problems are acceptable in a classroom setting, where children had time to familiarize themselves with the game, but they are less ideal for more informal learning settings like museums. We are hoping to rectify this problem in our next tutorial by introducing a tangible trigger that will not require children to touch the screen at all.

Figure 5. A player tapping to compile their constructed sequence.

Open World Environment

Our first iteration of the game had a linear narrative structure, consisting of a series of maze-like puzzles (Figure 6). Players puts down a sequence of blocks, pressed start, and watched the character navigate through the puzzle. If the character ran into water, the level would restart from the beginning. This style was heavily influenced by Hour of Code's Introduction to Code series [13]. But we found that this approach was not motivating or effective in open and collaborative spaces with multiple players on the same device. It was difficult for players to collaborate when there was only one answer to a problem. New players also had trouble understanding the game, as each new level used ideas introduced in previous levels.

For the next iteration, we moved away from the classic puzzle game, which tends to emphasize competitive and linear play with only one (or a small number) of correct answers [5]. We instead focused on developing a game which encouraged collaboration between players across different experience levels and ages by supporting different play-styles.

We decided instead to structure our game around a randomly generated open world environment, filled with interactable objects like strawberries, rats, cactuses, trees, water, animals, and more (Figure 6). We took out win/lose states, and instead introduced a more gradual build up of possessions that players could acquire. For example, finding seeds in the world allows you to grow strawberry plants in your farm. Making contact with undesired objects such as rats and cactuses sets Awbie off of a projected path, which deter players from further running into obstacles.

We hoped that an expansive world filled strawberries and other objects would encourage more open-ended and light-hearted collaboration. There are no wrong answers. We also wanted to encourage planning, but wanted to do so in a way that was less forced and more fun.

There was still structure in the game. We introduced lakes, with lilypads that lead to a treasure island in the middle (Figure 6). To get to the island, players must use a sequence of blocks that would guide the character all the way to the island. The character falls into the water if it takes a misstep and is transported back to the start of the lake. This gives players the option to explore more difficult puzzles, but they are also free to skip the lake and move on if they want.

We also introduced "Frenzy Mode", a fun animation that Awbie goes into when he is about to get multiple Strawbies with one sequence of blocks. This results in more Strawbies, a quicker movement speed, and playful animations. We felt that this was another example of adding structure into what is otherwise an open world game.

Evaluation

We noticed a big change in how players interacted with the game after we introduced the open-world environment. We heard a lot more conversations about decisions and goals, much like this excerpt from a conversation between two boys in Grade 2: P1: "Let's keep going. Wait actually I think I see strawberries up" P2: "No I think we should go down." P1: "Yeah but if we go up we can use Tornado!".

We also observed groups of children with different tendencies

play together. In one example, a boy and girl in Kindergarten are playing, and the boy continuously to Rainbow around the world. The girl says "See! We're not getting any Strawbies." After some discussion, the boy concedes, and the girl moves her sequence of blocks in front of the iPad.

Lakes were also effective. Many of our players understood that a more exact and elaborate sequence was necessary to reach the middle treasure island. Some players attempted to solve the puzzle, but others moved on to other goals. We saw the same behavior with Frenzy Mode. Many children only attempted chaining upon reaching a large obvious chain of Strawbies, while others attempted chaining at every possibility. Players showed signs of delight when they pulled off a successful chains.

7. CONCLUSION

We were able to create an inexpensive, portable, and fluid tangible programming environment building on the Osmo game system. Portability means that Strawbies can be played in a range of spaces, not tied to a camera fixture or interactive tabletop. Inexpensive and passive tangibles facilitate mass-production or print-and-play distribution scenarios for a broader audience. Fluidity helps create a more engaging experience that kept our players playing.

Through our long and iterative design process, we learned lessons that we believe are important for similar programming environments that combine real-time computer vision with inexpensive tangible compontents. Games should be designed using existing tablet interactions. Unlike a custom exhibit in a museum, iPads afford certain interactions that users will expect.

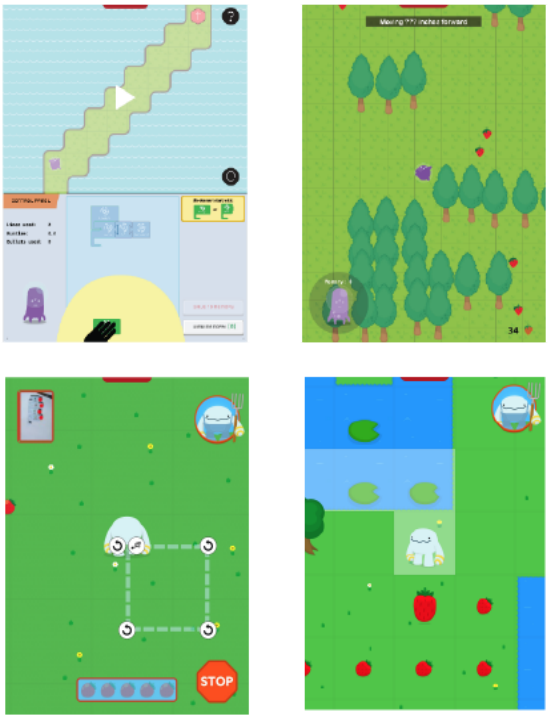

Figure 6. Different iterations with a puzzle-like environment (top left), a first pass at the open world environment (top right), preliminary real-time feedback (bottom left), and the current iteration (bottom right).

Games must be responsive. Passive tangibles are unable to provide 'smart' feedback, so the screen must be responsive to the player's actions in the physical space. A consistent response can be difficult to achieve using an Osmo, because hands will naturally block portions of the camera's vision. We've found that a consistent response can be achieved through fast image recognition, smart image analysis, and careful observation of how players move their hands. The lack of reliable, immediate, and consistent responses from the screen will be a roadblock that prohibits engagement.

8. ACKNOWLEDGEMENTS

We thank the administrative teams at Catherine Cook School, Francis W. Parker School, and Chicago's Children Museum for their support. We also thank all the children who have given us valuable feedback through testing. And we thank Eric Uchalik (www.euchalik.com) for designing the assets.

9. REFERENCES

[1] Cuban, L. (2009). *Oversold and underused: Computers in the classroom*. Harvard University Press.

[2] Chawla, K., Chiou, M., Sandes, A., & Blikstein, P. (2013). Dr. Wagon: a 'stretchable' toolkit for tangible computer programming. In *Proc. Interaction Design and Children* (IDC'13), 561-564.

[3] diSessa, A. (2000). *Changing Minds: Computers, Learning, and Literacy*. MIT Press.

[4] Flannery, L.P., Silverman, B., Kazakoff, E.R., Bers, M.U., Bontá, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proc. Interaction Design and Children* (IDC'13), 1-10.

[5] Flanagan, M. Critical play: radical game design. MIT press, 2009.

[6] Habgood, M.P.J., Ainsworth, S.E. Motivating children to learn effectively: Exploring the value of intrinsic integration in educational games. The Journal of the Learning Sciences 20.2 (2011): 169-206.

[7] Horn, M.S. (2013). The role of cultural forms in tangible interaction design. In *Proc. Tangible, Embedded, and Embodied Interaction* (TEI'13).

[8] Schweikardt, E., Gross, M. D. (2008). The robot is the program: interacting with roBlocks. In TEI'08. 167-168.

[9] Horn, M. TopCode: Tangible Object Placement Codes. http://users.eecs.northwestern.edu/~mhorn/topcodes/

[10] Horn, M.S., Crouser, R.J., & Bers, M.U. (2012). Tangible interaction and learning: The case for a hybrid approach. *Pers. and Ubiq. Computing, 16*(4), 379-389.

[10] Sullivan, A., Elkin, M., Bers, M. U. (2015). KIBO robot demo: engaging young children in programming and engineering. In Proceedings of the 14th International Conference on

Interaction Design and Children. 418-421.

[12] Horn, M.S., AlSulaiman, S., Koh, J. (2013). Translating Roberto to Omar: Computational literacy, stickerbooks, and cultural forms. In *Proc. IDC'13*, 120-127.

[13] Hour of Code. https://www.code.org/learn

[14] McNerney, T. (2004). From turtles to tangible programming bricks: explorations in physical language design. *Pers. and Ubiq. Computing, 8*(5), 326-337.

[15] Montemayor, J., Druin, A., Chipman, G., Farber, A., & Guha, M.L. (2004). Tools for children to create physical interactive StoryRooms. *Computers in Entertainment: Educating children through entertainment Part II, 2*(1).

[16] Nasir, N. I. S. (2005). Individual cognitive structuring and the sociocultural context: Strategy shifts in the game of dominoes. The Journal of the Learning Sciences, 14(1), 5-34.

[17] Norman, D.A. The design of everyday things: Revised and expanded edition. Basic books, 2013.

[18] Oh, H., Deshmane, A., Li, F., Han, J. Y., Stewart, M., Tsai, M., & Oakley, I. (2013). The digital dream lab: tabletop puzzle blocks for exploring programmatic concepts. In *Proc. Tangible, Embedded and Embodied Interaction* (TEI'13)

[19] Osmo. https://www.playosmo.com

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

[21] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Sliverman, B. & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

[22] Sapounidis, T., & Demetriadis, S. (2013). Tangible versus graphical user interfaces for robot programming: exploring cross-age children's preferences. *Personal and ubiquitous computing, 17*(8), 1775-1786.

[23] Schweikardt, E., & Gross, M.D. (2008). The robot is the program: interacting with roBlocks. In *Proc. Tangible and Embedded Interaction* (TEI'08), 167-168.

[24] Sipitakiat, A., & Nusen, N. (2012). Robo-Blocks: designing debugging abilities in a tangible programming system for early primary school children. In *Proc. Interaction Design and Children* (IDC'12), 98-105.

[25] Snibbe, S. S., Raffle, H. S. (2009). Social immersive media: pursuing best practices for multi-user interactive camera/projector exhibits. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1447-1456.

[26] Stevens, R., Satwicz, T., & McCarthy, L. (2008). In-game, in-room, in-world: Reconnecting video game play to the rest of kids' lives. The ecology of games: Connecting youth, games, and learning, 9, 41-66.

[27] Weintrop, D., & Wilensky, U. (2013). RoboBuilder: A Computational Thinking Game. In *Proc. ACM Technical Symposium on Computer Science Education*, 736-736.

[28] Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *Journal of the Learning Sciences, 17*(4), 517-550.