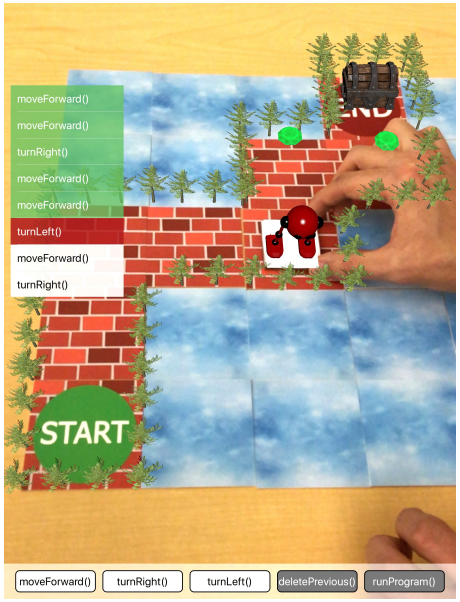


Figure 1: A user using our mixed-reality coding learning platform to physically interact with a path-finding program and debug it, by moving an avatar tile (red virtual avatar anchored on top). Our system tracks the avatar's movement in real time (with image tracking), to give immediate visual feedback. Correct and incorrect code blocks are colored green and red respectively (on the left). The game board (i.e., programming problems) are configurable, so are all virtual objects (e.g., trees lining path, motivational green gems and treasure chest).

Mixed Reality for Learning Programming

Joonyoung Kim
Georgia Institute of Technology
Atlanta, GA, USA
jkim936@gatech.edu

Sudeep Agarwal
Georgia Institute of Technology
Atlanta, GA, USA
sagarwal88@gatech.edu

Kristina Marotta
Georgia Institute of Technology
Atlanta, GA, USA
kmarotta3@gatech.edu

Siwei Li
Georgia Institute of Technology
Atlanta, GA, USA
robertsiweili@gatech.edu

Jonathan Leo
Georgia Institute of Technology
Atlanta, GA, USA
jleo7@gatech.edu

Duen Horng Chau
Georgia Institute of Technology
Atlanta, GA, USA
polo@gatech.edu

ABSTRACT

We present our ongoing investigation into leveraging mixed reality (MR) to help students learn coding more easily and with more fun. We have developed an MR coding learning platform using Apple's ARKit 2 on iOS, with a physical user-configurable coding game board. Our approach could provide major benefits over conventional augmented reality (AR) approaches for learning coding and debugging: (1) allowing teachers to tailor the platform to their instructional needs, and spark creativity and engagement among students in designing programming problems that interest them; (2) enabling students to physically interact with a program, concretizing coding errors and providing real-time visual feedback to aid students' program understanding and reduce cognitive load. We

IDC'19, June 2019, Boise, US

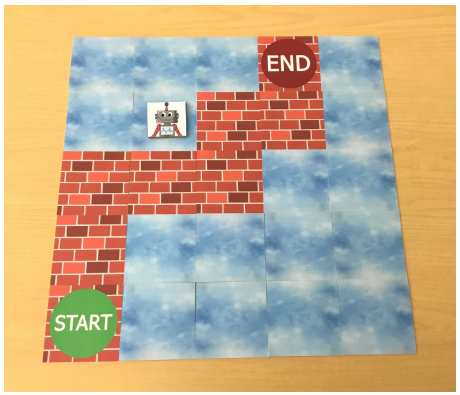
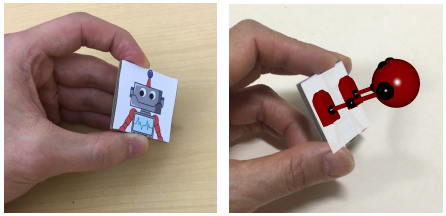


Figure 2: Physical user-configurable game board, with brick *path* tiles, *start* and *end* tile, and an *avatar* tile. Cloud tiles fill empty space to keep path tiles in place.



(a) Physical tile **(b) Avatar anchored**

Figure 3: ARKit's *image tracking* tracks the image on (a) physical tile and (b) anchors virtual avatar on it in real time.

present our preliminary results that uses ARKit's *image tracking* and *object detection* to enable core mixed-reality interaction capabilities on our platform.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**; • **Human-centered computing** → *Mixed / augmented reality*;

KEYWORDS

Mixed reality, augmented reality, programming education, coding, ARKit

ACM Reference format:
Joonyoung Kim, Sudeep Agarwal, Kristina Marotta, Siwei Li, Jonathan Leo, and Duen Horng Chau. 2019. Mixed Reality for Learning Programming. In *Proceedings of ACM IDC conference, Boise, US, June 2019 (IDC'19),* 6 pages. https://doi.org/10.475/123_4

INTRODUCTION

Rapid advances in augmented reality (AR) technologies have created new education possibilities, from visualizing abstract concepts and scientific phenomena [7], to improving laboratory skills [1]. Our recent work [3] demonstrated that AR's benefits could extend to helping students learn coding more easily and with more fun. However, AR also creates new challenges for students and educators. Some studies reported that AR systems often cause cognitive overload [2, 4]. AR systems can be difficult for teachers to adopt to meet their instructional objectives, as the AR content is often fixed [7].

We present our ongoing investigation into leveraging **mixed reality (MR)** to address some of the above challenges and to enhance the learning experiences for coding. To evaluate our ideas' feasibility, we have developed an MR coding learning platform (Fig. 1) using Apple's ARKit 2 on iOS, with a physical user-configurable coding game board, and a tangible avatar object that would navigate towards a goal on the board. This research builds on our earlier AR work [3]. But unlike conventional AR, which primarily overlays virtual objects onto the real world and the user would still interact with them digitally, our MR approach for coding aims to provide major education benefits:

(1) **Physical avatar to concretize coding errors.** We leverage MR to introduce a novel way for students to *physically* interact with the program execution process and to debug it step by step (see Fig. 1). Students would "trace" a program's code by *physically* moving a physical avatar object. Our system tracks the student's movements in real time and give immediate visual feedback, which could help reduce their cognitive load [7] and improve their understanding of the errors. Debugging is a crucial skill that can greatly improve students' understanding of their programs [5, 6].

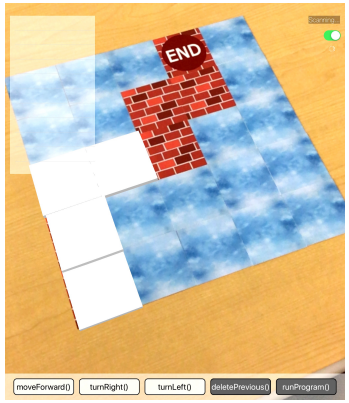


Figure 4: Scanning physical path tiles ("bricks") into a virtual path, using image detection. Successfully scanned tiles are temporarily highlighted in white.

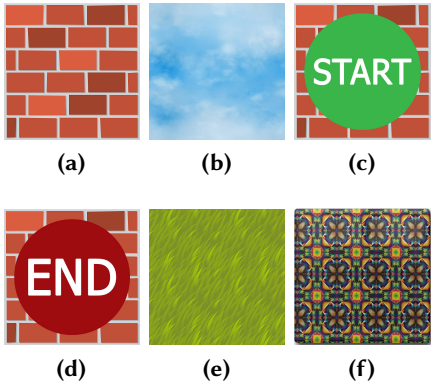


Figure 5: Board Tiles: (a) *path* tile, (b) *filler* tile, (c) *start* tile, and (d) *end* tile. *Start* and *end* tiles are starting and end point of the path. Example "bad" tiles: (e) low contrast; (f) disorienting pattern.

(2) **Tailored game boards.** A main pedagogical challenge that educators face in using AR systems to accomplish instructional objectives lies in the inflexibility of the system content. Our approach could help teachers more easily tailor the platform to their instructional needs [7], and spark creativity and engagement among students (e.g., design coding problems, virtual avatars).

CODE LEARNING PLATFORM

Using our code learning platform, the user can write a *path finding* program that builds a path to guide an avatar to a goal. The user composes the program using commands shown at the bottom of the user interface (see Fig. 1): *turn left*, *turn right*, and *move forward*.

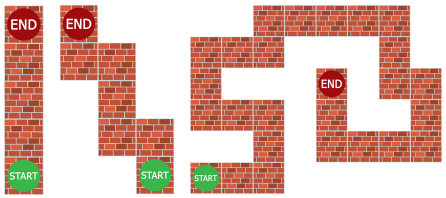
To execute the code — different from most other systems for learning programming — we leverage MR to introduce a novel way that engages the user to *physically* evaluate and debug their program. Figure 1 shows an example, where the user "traces" the written program, by *physically* moving the physical avatar tile. Our system tracks the user's movements in real time and gives immediate visual feedback as to whether they match the (virtual) program written. This debugging-like process allows the user to gain a better understanding of coding errors that they may have made, helping to reduce their cognitive load [7].

Tailored Game Board. Through different arrangements of physical tiles to create the game board (as in Fig. 2), users can construct their own learning experience, which offers more flexibility for teachers to tailor their course content compared to existing AR applications [7]. This is done using image detection in ARKit, which allows us to identify the position of the tile placements and accordingly generate a virtual environment around the tiles (Fig. 1).

To build the game board, the user builds a grid using *brick* tiles (Fig. 5a) and *filler* tiles (Fig. 5b). The filler tiles are used to surround the brick path, forming a square or rectangular grid. The contrasting color between the two types of tiles helps with image detection and makes the board visually appealing. A wide range of path complexity can be achieved using these tiles (Fig. 6).

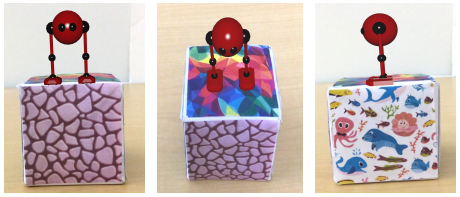
Avatar. After the user completes the coding task, the user is able to move a physical avatar object to simulate the execution using image tracking. This allows the user to move the physical tile around and see a virtual character following the direction of the tile in real-time (Fig. 3).

Interface Design. The user interface brings the game board and avatar together, allowing the user to solve the path finding task. When the interface is launched, the user is asked to scan the tiles in order to build a map of the game board (Fig. 4). The user then places the physical avatar on the starting tile, and begins to plan the path using *moveForward()*, *turnLeft()*, and *turnRight()* buttons within the interface. Each corresponding code block is then added to the left-hand side panel. When the user is ready to execute their program, they select the *runProgram()* button, which highlights the first code block in the panel orange, prompting the user to perform the action. If the performed action leads to



(a) (b) (c)

Figure 6: Paths with varying complexity can be created for learning different programming concepts (e.g., for loops, if statements), from (a–b) simpler paths to (c) long, complex paths with many turns.



(a) Front (b) Front 45° (c) Side

Figure 7: "3D" image tracking of avatar cube from (a) front, (b) 45 degrees angle from front, and (c) side.

a correct step in the solution, the highlighted block turns green. However, if the user performs the wrong action, or the planned step is incorrect, the block turns red, prompting the user to correct the step (Fig. 1). This process is repeated with each step until the user manages to bring the avatar to the destination tile, or reaches the end of the planned path.

IMPLEMENTATION

We implemented the above system using Apple's ARKit 2, which provided us with a real-time image tracking implementation and, more importantly, the *world tracking* capability that establishes a correspondence between *real* and virtual *spaces* — a crucial feature required for the system to scan the user-designed paths and to "anchor" the paths in the physical world to enable our novel learning approach through MR. World tracking is not available in other popular AR SDKs such as Vuforia. Below, we described details of our implementation and how it resolved key technical challenges.

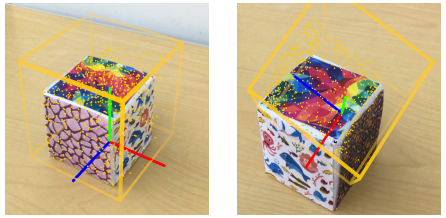
Game Board Detection. We created 2.56" × 2.56" square tiles, on which a brick image is printed. Three important characteristics were used to choose an appropriate image for the path tiles:

(1) **High Resolution**: 512 × 512 pixels or better
(2) **High Contrast**: sharp lines and contrasting colors
(3) **Wide, Flat Histogram**: variety and even distribution of colors

The brick image chosen for our tiles pictured in Figure 5a, has higher contrast and a more appropriate histogram than that in Figure 5e. The sharp lines throughout the pattern in our chosen image aids in detection, maximizing the efficiency of ARKit's recognition of the board. While the tile shown in Figure 5f satisfies all three characteristics, it would become hard on the user's eyes in a path. As such, tile in Figure 5a was chosen.

Avatar Tracking. Throughout the process of tracking the game's avatar tile, both the *distance* and *angle* at which the device camera is held with respect to the avatar affect its detection speed and accuracy, ultimately impacting the user experience. Tracking may drop off at longer distances or at smaller angle of depression between the camera and avatar. To "re-track" the avatar tile, ARKit requires the user to first move the device camera significantly closer to the avatar tile to "lock it in" before moving the camera further away. We hypothesized that instead of using a flat 2D avatar **tile** (2.25" × 2.25"), using a **cube** (2.25" on all sides) with different images on its 6 faces would provide more opportunities for ARKit to track (at different angles and distances), thus improving the tracking robustness. To better understand the pros and cons of these two physical designs (tile vs. cube), we tested them against ARKit's built-in *image tracking* and *object detection* functionality. Specifically, we tested:

(1) **2D Image Tracking**: image tracking of a single avatar tile image
(2) **"3D" Image Tracking**: simultaneous image tracking of all 6 images of the cube



(a) Correct orientation **(b) Incorrect**

Figure 8: 3D object detection of avatar cube. (a) Virtual orientation matches physical orientation. (b) When detected from some angles, virtual orientation does not match physical counterpart.

Table 1: The farthest distances from the physical avatar (tile or cube) in centimeters for which the device may be held, at angles of depression of 90°, 60°, 45°, 30° and 0°.

Physical Avatar	Tracking Method	90°	60°	45°	30°	0°
Tile	2D Image Tracking	96.54cm	73.62cm	58.34cm	44.57cm	N/A
Cube	"3D" Image Tracking	104.68cm	78.33cm	63.41cm	41.88cm	65.86cm
	3D Object Detection	64.59cm	71.47cm	77.84cm	68.44cm	55.89cm

(3) **3D Object Detection**: scanning for and detecting the avatar cube, on an interval timer, emulating object "tracking"

We evaluated which approach would be the most optimal to use in our system and summarized our findings in Table 1. For the avatar tile, the angle of depression and distance were measured from the midpoint of the avatar (laid flat) to the device camera. For the cube, they were measured from its center.

For the second approach ("3D" Image Tracking), we simultaneously tracked all 6 sides of the cube (using same 2D image tracking in our first approach), while displaying the virtual avatar on top of the cube regardless of which side it was detected from (see Fig. 7a–c). Tracking all sides provided redundancy. If tracking was lost on the top face image, it could be picked back up from one of the images on other sides or vice versa. While this approach allowed tracking at low angles of depression (and our first method did not), tracking still failed at some angles. The biggest drawback of this approach is the large size of the cube, which blocks much of the user's view of the game board, both physically and virtually. Our third approach (3D Object Detection) allowed the avatar to be recognized at most angles. However, since ARKit 2 only supports 3D object detection and not continuous tracking, the avatar was not recognized in real time. We also found that ARKit occasionally fails to correctly detect the avatar cube's orientation (like in Fig. 8b), thus incorrectly orientating the virtual avatar, decreasing usability. Comparing all three approaches, we decided the first one was the most efficient, optimal mechanism for our system.

ONGOING WORK

User Testing. In order to examine whether mixed reality, specifically the hybrid use of physical and virtual objects, would enhance the learning experience, we plan to conduct lab studies to evaluate six areas of usability and effectiveness of our approach: immersiveness, ease of debugging, ease of planning, likeability, ease of use, and task completion time. We plan to compare these results with those from our earlier AR-based work [3] that did not evaluate or support any hybrid physical-virtual

interaction, to assess the value added by the mixed reality. Since this project has been mainly targeting beginning learners of coding, we plan to recruit participants from middle school and high school who have little or no previous programming experience. Through discussion and consultation with Georgia Tech's *Center for Education Integrating Science, Mathematics and Computing* (CEISMC), we have identified a local high school, Grady High School, for our first engagement.

We are also working to support more programming concepts (e.g., loops and conditionals), and implementing corresponding path finding challenges that stimulate students' learning interest.

CONCLUSIONS

We presented our ongoing investigation into leveraging mixed reality (MR) to help students learn coding more easily and with more fun. we have created an MR coding learning platform using Apple's ARKit 2 on iOS, with a physical user-configurable coding game board, and a tangible avatar object that would navigate towards a goal on the board. We believe our MR approach offers major benefits over conventional augmented reality (AR) approaches: (1) allowing teachers to tailor our platform to their instructional needs, and spark creativity and engagement among students in coming with program problems that interest them; (2) enabling users to physically interact with a program, concretizing coding errors and providing real-time visual feedback that could aid users' understanding of the program and lessen cognitive load. As we proceed with our planned user studies, we will better understand the potential of mixed reality in enhancing beginners' learning experience for coding.

REFERENCES

[1] Murat Akçayır, Gökçe Akçayır, Hüseyin Miraç Pektaş, and Mehmet Akif Ocak. 2016. Augmented reality in science laboratories: The effects of augmented reality on university students' laboratory skills and attitudes toward science laboratories. *Computers in Human Behavior* 57 (2016), 334–342.

[2] Kun-Hung Cheng and Chin-Chung Tsai. 2013. Affordances of Augmented Reality in Science Learning: Suggestions for Future Research. *Journal of Science Education and Technology* 22, 4 (01 Aug 2013), 449–462. https://doi.org/10.1007/s10956-012-9405-9

[3] Nathan Dass, Joonyoung Kim, Sam Ford, and Sudeep Agarwal. 2018. Augmenting Coding : Augmented Reality for Learning Programming. (2018), 156–159. https://doi.org/10.1145/3202667.3202695

[4] Matt Dunleavy, Chris Dede, and Rebecca Mitchell. 2009. Affordances and Limitations of Immersive Participatory Augmented Reality Simulations for Teaching and Learning. *Journal of Science Education and Technology* 18, 1 (01 Feb 2009), 7–22. https://doi.org/10.1007/s10956-008-9119-1

[5] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. 2014. Principles of a debugging-first puzzle game for computing education. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*. IEEE, 57–64.

[6] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Computer Science Education* 18, 2 (2008), 67–92.

[7] Hsin-Kai Wu, Silvia Wen-Yu Lee, Hsin-Yi Chang, and Jyh-Chong Liang. 2013. Current status, opportunities and challenges of augmented reality in education. *Computers & education* 62 (2013), 41–49.