App Inventor VR Editor for Computational Thinking

Jane IM 1, Paul MEDLOCK-WALTON2*, Mike TISSENBAUM2*

1 Korea University
2 Massachusetts Institute of Technology
jane605@korea.ac.kr, paulmw@mit.edu, mtissen@mit.edu

ABSTRACT
This paper introduces the concept of a virtual reality (VR) programming environment that allows youth to both develop immersive VR experiences while enhancing computational thinking (CT). Specifically, we extended a blocks-based programming platform, MIT App Inventor, to allow youth to make VR Android apps (AI/VR). We compare AI/VR's support for CT to other existing VR editors using the CT concepts established by Brennan and Resnick (2012). Comparisons showed that AI/VR’s support for all CT concepts and its ease of use for kids, makes it more preferable for teaching CT compared to other editors.

KEYWORDS
computational thinking, virtual reality, constructionism, immersive interface, MIT App Inventor

1. INTRODUCTION
In recent years, many educators have argued computational thinking (CT) (Wing, 2006) is an indispensable skill for everyone. In order to support widespread uptake of computational thinking, blocks-based approaches to programming have been developed, in which users program by snapping blocks of code. For example, Scratch allows students to build 2D multimedia (Brennan & Resnick, 2012). Alice helps students learn programming by building 3D media (Dann, Cooper, & Pausch, 2006).

Compared to Scratch, Alice provided a more immersive experience. Studies using Alice showed it is effective for learning programming, in part to its immersive nature (Sykes, 2007), indicating the potential for immersive experiences to enhance students’ computational thinking. However, there has been limited research on how VR, an immersive environment, can support CT learning. Given the nascent field of VR, in order to understand the role of it in developing CT, there is a need to examine current VR editors. If these do not support the kinds of learning we wish to support, then it is critical to develop appropriate tools. This work was framed around two needs: 1) understand the state of current VR editors and examine their suitability for supporting developing computational thinking; and 2) develop a tool that supports the learning needed, if current platforms were found to be lacking.

Below, we examine current VR editors, discuss how they support CT and their suitability for young learners. We propose AI/VR that responds to their shortcomings.

2. BACKGROUND

2.1. Constructionism
Constructionism is the process of building understanding through the active use of tools to develop tangible artifacts (Kafai & Resnick, 2011). Building on constructionism, is the concept of “learning as designers”, which has shown to increase higher-order thought process development and motivation (Cooper, Dann, & Pausch 2003; Fortus, Dershimer, Krajcik, Marx, & Mamlok-Naaman, 2004). Especially, programming interactive media has been shown to support CT (Brennan & Resnick, 2012). The development of interactive and immersive media, with platforms such as Alice, also embody constructionist characteristics, as they allow learners to design interactive media freely in the same context (Sykes, 2007).

2.2. Immersive Interface for Learning
Immersion is the subjective impression that one is participating in a comprehensive, realistic experience (Stanney, 2002; Lessiter, Freeman, Keogh, & Davidoff, 2001). Studies have shown that immersion in a digital environment can enhance education in at least three ways: allowing multiple perspectives, situating the learning, and transfer to other contexts (Dede, 2009). Below we describe two immersive learning environments.

2.3. Alice
Alice is a 3D graphics programming environment that allows users to create interactive 3D animations and learn programming in an object-oriented approach (Dann et al., 2006). Research on use of Alice to teach an entry level undergraduate computer science course showed that posttest performance among students who used Alice was significantly higher than comparison groups. Qualitative results showed that students using Alice enjoyed the process and spent more time engaged in the course (Sykes, 2007). There were diverse reasons for this engagement, including

the active graphical interface. While Alice is not completely immersive, it has a higher degree of immersion compared to text-based languages. The results suggest the potential for enhancing students’ computational thinking skills within a more immersive environment.

2.4. Immersive Interface for Learning
Virtual Environment Interactions (VEnvI) is a platform that uses a database of dance sequences, VR, and a drag-and-drop interface to teach programming concepts
Although this study is limited because students did little programming, and were introduced to programming concepts in sessions, results showed that students found the immersion of VEnVI desirable and became more positive towards computer science.

3. PREVIOUS EDITORS

3.1. Introduction of VR Editors


3.2. Categorization and Analysis

We categorized the editors using Brennan and Resnick’s computational concepts, which are sequences, loops, parallelism, events, conditionals, operators, and data (Brennan & Resnick, 2012). The editors are also categorized based on their affordances into three groups, which are photo/video focused editors, visual programming editors, and text based editors (Table 1).

3.2.1. Photo or video focused editors

Photo or video focused editors are ones that: 1) focus on making rich scenes using photos and videos; and 2) only support acquiring the computational thinking concept ‘events’. The focus on scene creation is the goal of these editors, which explains why they have limited utility for CT. Users can place events inside scenes using drag-and-drop (e.g., adding a button that is clickable). However, such editors lack the means to use sequences, loops, parallelism, events, operators, and data. 360°& VR Editor, HoloBuilder, and Smart2VR fit in this category.

3.2.2. Visual programming editors

Visual programming editors support most, if not all, the CT concepts through visual programming. Vizor and CoSpaces fall in this category, with CoSpaces also supporting text based programming.

In Vizor, students can apply all computational thinking concepts through using a ‘patch’, which can be connected to other patches (Figure 1). Users can combine patches like state/structure patches to understand sequence. There are prebuilt patches for loops, conditionals, operators, variables, and data. Users can learn parallelism, for example, by using two mouse press patches. However, there are limitations including the limited animation patches, which can make animating objects difficult. Additionally, unlike MIT App Inventor where blocks run from top to bottom, the order of patches do not indicate sequence in Vizor, requiring users to link extra patches.

In CoSpaces, users can use blocks to apply all seven CT concepts. Users can connect blocks from top to bottom to understand sequence, and use loop blocks to understand loops. The execute in parallel and the on activate of blocks enable parallelism and events in users’ projects, respectively. There are prebuilt blocks for conditionals, operators, variables and data. However, CoSpaces lacks blocks for dynamically creating objects, and has limited types of events compared to Vizor and AI/VR. These can limit the range of computational practices (which focus on “how”, instead of “what” users learn) (Brennan & Resnick, 2012).

Table 1. Categorization of VR Editors with colored boxes representing an attribute(column) that an editor(row) has.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Editor Type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Does not require programming background</th>
<th>Intended audience</th>
</tr>
</thead>
<tbody>
<tr>
<td>360°&amp; VR Editor</td>
<td>Photo/Video</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Novices, especially construction company</td>
</tr>
<tr>
<td>HoloBuilder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Novices</td>
</tr>
<tr>
<td>Smart2VR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Novices</td>
</tr>
<tr>
<td>Vizor</td>
<td>Visual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Novices</td>
</tr>
<tr>
<td>CoSpaces</td>
<td>Visual/Text based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Novices</td>
</tr>
<tr>
<td>Unity</td>
<td>Text based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Professional, Experienced gamers</td>
</tr>
<tr>
<td>Unreal Engine VR Editor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Professional, Experienced gamers</td>
</tr>
<tr>
<td>Arma 3’s editor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Professional, Experienced gamers</td>
</tr>
<tr>
<td>Simmetri</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Artists</td>
</tr>
<tr>
<td>AI/VR</td>
<td>Visual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Novices</td>
</tr>
</tbody>
</table>
3.3. Text-based editors
Text based editors are editors that require at least partial text based programming to exhibit the seven CT concepts. Unity, Unreal Engine VR Editor, Arma 3’s virtual reality editor, and Simmetri fit in this category, with differing audiences (Table 1). Since these editors allow users to build complicated VR environments technically, they support all CT concepts, and users can employ complex CT with them. However, these text-based editors require a steep learning curve and are not suitable for beginners.

Building off of the various shortcomings of the tools described above, we identified a gap in the VR authoring landscape for a tool that allows novices to develop VR applications while developing CT concept understandings. Below we describe the tool and its use.

4. APP INVENTOR VR EDITOR
4.1. Blocks in AI/VR
There are four kinds of blocks in AI/VR: 1) event, 2) method, 3) property setter and getter, and 4) object creation blocks. Event includes checkButton, which checks if the user clicks the Cardboard button. Method blocks trigger interactions with objects and the player, including moveUser, which changes the location of the player. Property setters and getters change object’s attributes, such as size. Object creation blocks, such as createCube, allow the user to dynamically add objects.

4.2. Sample AI/VR program
To demonstrate how the editor supports CT, we included a sample AI/VR program (Figure 2 & Figure 3). If the user gazes at one of the four cubes (that are made by shaking the phone), she will earn points (shown in a label). The cube also moves to a random position and changes color.

This example shows how all seven computational concepts are supported in AI/VR. First, users can understand sequences by checking that blocks are executed in order when the user gazes at a cube. The cube changing color, and the score increasing and updating shows sequentially ((1) of Figure 3). Loops are used to iterate over cubes in (2). Events are used through blocks like checkGazeShort (the block that returns 1 if the user gazed at the object - (3)). For parallelism, two "if" blocks are used to check whether a cube was gazed at and the color and position of the cube are changed concurrently ((4)). For conditionals, the user can connect event blocks and attribute or animation related blocks with an "if" block ((5)). For operators, users can practice addition by adding 1 to the current score when a cube is gazed at (6) and checking that the increased score is updated in the scene (6-2). Lastly, users can understand data by keeping track of cubes using a list block ((7)).
a limitation because it could reduce the diversity of computational practices.

5. CONCLUSION
Considering the role constructionism plays in computational thinking and the possibility of immersive virtual reality to support this learning, we introduce AI/VR - a blocks-based tool to support kids to more easily create virtual reality apps. We have shown AI/VR’s fit as a visual programming editor that supports all seven of Brennan and Resnick’s CT concepts, while also being usable by young kids. Although AI/VR has limited diversity in objects and media related blocks, it overcomes the limitations of Vizor, such as animating objects and sequence, and those of CoSpaces such as the lack of blocks for dynamically creating objects and the limited types of triggers for events.

6. ACKNOWLEDGEMENT
We would like to thank Hal Abelson, Professor of EECS at MIT, whose insight was great help to this research.

7. REFERENCES


