*electronics*

MDPI

*Article*

# BlocklyAR: A Visual Programming Interface for Creating Augmented Reality Experiences

**Vinh T. Nguyen [1,†], Kwanghee Jung [2,*,†] and Tommy Dang [1]**

[1] Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA; vinh.nguyen@ttu.edu (V.T.N.); tommy.dang@ttu.edu (T.D.)

[2] Department of Educational Psychology and Leadership, Texas Tech University, Lubbock, TX 79409, USA

* Correspondence: kwanghee.jung@ttu.edu

† These authors contributed equally to this work.

Received: 1 July 2020; Accepted: 24 July 2020; Published: 27 July 2020


**Abstract:** State-of-the-art tools for creating augmented reality (AR) applications often depend on a specific programming language and the deployed target devices. The typing syntax of a program is error-prone, and device dependency makes it difficult to share newly created AR applications. This paper presents BlocklyAR, a novel web-based visual programming interface for creating and generating an AR application. This tool is intended for non-programmers (young learners and enthusiasts) who are interested in making an AR application. The goals of this tool are: (1) to help young learners and enthusiasts express their programming ideas without memorizing syntax, (2) to enable users to perceive their expressions, (3) to enable learners to generate an AR application with minimal effort, and (4) to support users by allowing them to share newly created AR applications with others. BlocklyAR uses Blockly for creating a palette of commands and AR.js for transcribing commands into AR experience. The applicability of BlocklyAR was demonstrated through a use case where an existing AR application was recreated by using our tool. The result showed that our tool could yield an equivalent product. We evaluated the visual tool with the help of 66 users to gather perspectives on the specific benefits of employing BlocklyAR in producing an AR application. The technology acceptance model was adapted to assess an individual's acceptance of information technology.

**Keywords:** A-Frame; Blockly; augmented reality; storytelling; visual programming interface; STEM/CS education; technology acceptance model; generalized structured component analysis

## 1. Introduction

Augmented reality (AR) is a technology that expands the physical world with additional digital information such as sounds, images, and models [1]. The central value of AR is that the components of the digital world blend into a person's perception of the real world. Beyond showing the data, it allows the integration of immersive sensations, which are perceived as natural parts of an environment. In recent years, the growth of AR applications can be attributed to solutions that focus on contextualizing information (e.g., annotating different parts of a physical object [2], displaying artifacts at a given place [3], and aligning virtual objects with the real world [4]—that is, automatically positioning an object on the detected table or floor). In an educational setting, AR technology can be incorporated in the classroom to enhance teaching/learning efficiency and the motivation of both educators and learners [5,6].

There are many available tools and libraries—such as ARCore, ARKit, Vuforia, Unity [7], and ARToolkit [8]—that developers can use to create and manipulate AR applications. ARCore is a

framework from Google for building augmented reality applications for both Android and iOS devices. Apple provides ARKit for making AR applications and games for their iOS devices. Both of these frameworks use a handheld device's sensors for motion tracking, light estimation, and environmental understanding. Unlike ARCore and ARKit, ARToolkit and Vuforia are computer tracking libraries that overlay virtual objects on the real world based on markers. The above frameworks and libraries can be integrated in Unity for porting an AR application in devices with different operating systems (e.g., Android or iOS). To use these frameworks, it is necessary to have an integrated development environment (e.g., Xcode for iOS and Android Studio for Android, or Unity for both iOS and Android), a compatible device (e.g., ARCore requires that the device must be running Android at least 7.0, and ARKit requires device to be run on iOS 11 with an A9, A10, or A11 processor), and knowledge of a specific programming language (e.g., Objective-C, C#, C/C++, or Java). These requirements are obstacles for beginners who want to create an AR experience on their own. In their study, Nguyen et al. [9] showed that API version incompatibility in the integrated development environment (IDE) is a major obstacle students face while working with the application in terms of both coding and deploying. The study also indicated that students faced difficulty in analyzing "800+ line scripts".

To alleviate the difficulty of having an IDE and a compatible device, web-based AR (e.g., WebVR or WebXR) is an alternative approach for users to experience virtual objects in the real world only by using a web browser on their handheld device. A web browser engine supports Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and scripting languages such as JavaScript that can be programmed with a simple text editor, thereby releasing users from the need for an IDE. Furthermore, web-based AR toolkits such as ARToolkit for web [8] are compatible with a wider range of devices (e.g., running on OS 4.0.3 or higher for Android and 7.0 or higher for iOS). The use of ARToolKit has been widely adapted in other libraries such as ThreeJS [10] and A-Frame.io [11]. Currently, although web-based AR has some limitations due to its low frame-rate and not leveraging the full capacity of in-application AR, the continual development of other technologies will help web-based AR keep growing in the future. For example, the presence of an open binary instruction format WebAssembly [12] would allow the browser engine to run native code in a browser, and this provides the capability to access the in-application AR features through a web-based VR.

Recent research has produced some insights that describe how to lessen the issue of mastering a certain programming language for young learners and enthusiasts. Block-based programming [13] is a type of programming language where instructions are mainly represented as blocks (or visual cues) and users drag and drop the cues to form a set of instructions. This programming paradigm enables developers to focus on logical programming rather than memorizing the syntax of coding. Scratch [14] is an example of using this visual paradigm in K-12 education. By using Scratch, K-12 students are able to program a 2D game and experience it immediately on a web-browser. Radu and Blair [15] extended Scratch to make it possible to use to create AR. However, their work stopped at rendering 2D images on the screen only; 3D objects and spatial position manipulation have not been implemented. CoSpaces [16] is a commercial product created for education; it enables students to create virtual reality (VR) applications through context-based language. It also can be used to create a simple AR application by superimposing 3D objects onto the physical environment. However, the interactions between 3D objects are limited. In addition, Laine [17] pointed out that the majority of AR applications were developed through Vuforia SDK with a few occurrences of ARToolkit, which would not be suitable for non-programmers, but expert programmers. Furthermore, Wu et al. [18] indicated that in some AR systems, the learning content and the teaching sequence are rather "fixed" such that teachers are not able to make changes to accommodate students' learning needs. As such, an authoring/storytelling tool is highly needed for teachers and students to create AR applications [19].

## 2. Motivation and Research Aim

Motivated by the block-based visual programming paradigm and AR for the web, we aim to bridge the gap between existing technologies. Our intention is to provide a generic web environment

and let users to freely create AR applications for their interests. To the best of our knowledge, there is no tool in the literature that offers these features, which makes this research a unique contribution. To address this gap, this paper introduces BlocklyAR, a novel visual programming interface for creating and generating a web-based AR application. The goal of our work is to help learners create and use AR without having to memorize coding syntax. Consequently, this paper contributes to current research as it:

- Helps young learners and enthusiasts express their programming ideas without memorizing syntax;
- Enables learners to evaluate their coding promptly;
- Allows learners to generate an AR application with minimal effort;
- Supports users' ability to share newly created AR applications with others;
- Shows the applicability of BlocklyAR by recreating an existing AR application through a visual programming interface;
- Evaluates BlocklyAR tool using the technology acceptance model with the following hypotheses:

**Hypothesis 1 (H1).** *Perceived Visual Design positively influences perceived task–technology fit.*

**Hypothesis 2 (H2).** *Perceived task–technology fit positively influences perceived ease-of-use.*

**Hypothesis 3 (H3).** *Perceived visual design positively influences Perceived usefulness.*

**Hypothesis 4 (H4).** *Perceived ease-of-use positively influences Perceived usefulness.*

**Hypothesis 5 (H5).** *Perceived ease-of-use positively influences intention to use.*

**Hypothesis 6 (H6).** *Perceived usefulness positively influences intention to use.*

The rest of this paper is organized as follows: Section 3 summarizes existing research that is close to our paper. Section 4 presents methods for constructing our proposed tool and describes the tool's architecture in detail. Section 5 evaluates the tool's application using the technology acceptance model. Some challenging issues are discussed in Section 6. Our paper is concluded in Section 7.

## 3. Related Work

In this section, we briefly review the use of block-based visual programming, using Blockly [20] as an example. While Blockly has been used in many different domains, we only discuss the research that is relevant to our study, particularly in making AR applications.

Blockly [20] is a client-side library, a project of Google, for creating block-based visual programming languages and editors (as depicted in Figure 1). One interesting feature of Blockly is that it can run in a web browser. Visual cues in Blockly can be linked together to make writing code easier. The central value of Blockly is the ability to generate code in many different languages, such as JavaScript, Lua, Dart, Python, and PHP. A typical application that uses Blockly is Scratch [14]. This web-based visual programming language is targeted primarily at children who are between the ages of 8 and 16.

Radu and Blair [15] developed a first AR Scratch tool that allows children to create programs that mix real and virtual spaces. In their work, they customized the Scratch environment by adding an AR feature to the interface. ARToolkitPlus library was used for detecting markers' position and orientation. Once the markers were found, 2D actor sprites (or images) were overlaid on to the corresponding markers. The pilot study result showed that young learners were enthusiastic, and returned to interact with the tool even after the study finished. As indicated by the authors, the main drawback of this

tool was not adding a third dimension to the Scratch environment due to its complexity in specifying relationships and interactions between objects.

Mota et al. [21] built an in-application AR tool called VEDILS (which stands for Visual Environment for Designing Interactive Learning Scenarios). Android users can leverage this tool in order to create AR applications. The VEDILS's AR components were developed using the Vuforia for image recognition and tracking. Like Scratch, VEDILS relied on the Blockly library for generating visual blocks.

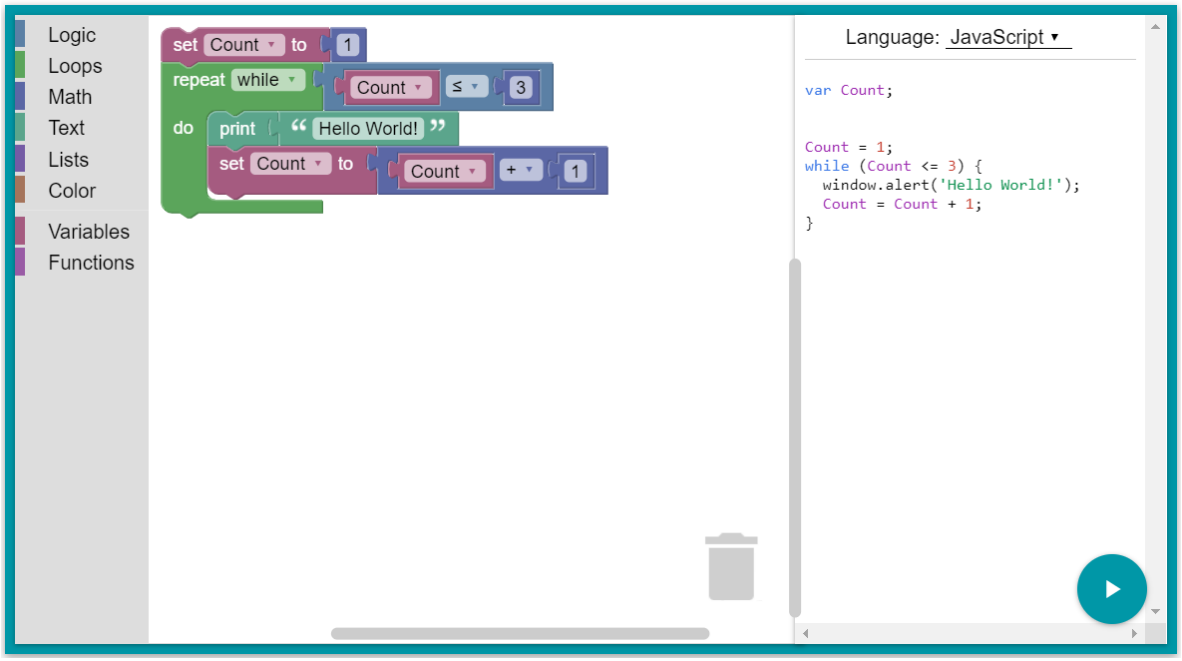


**Figure 1.** The visual interface of Blockly for generating JavaScript from blocks.

The idea most similar to our work in the literature was presented recently by Clarke [22], in which the author extended VEDILS for working on iOS devices. His tool enables users to work with 20 augmented reality primitive components, including basic shapes such as boxes, capsules, cones, spheres, text, and 3D models. A tutorial for using the tool was provided to help participants get familiar with the AR components and the visual interface. The pilot study result showed that participants felt empowered by working with the AR components and they could build AR applications after using the AR components. As the author pointed out, API incompatibility was one of the main issues in the study since the tool required iOS 12+ to run. In addition, features such as animations and movements in the AR environment had not developed and these components were put in the future work.

Although there are still a number of tools in the literature for generating AR applications, they are out of the scope of this paper since we are focusing on block-based programming language. The aforementioned studies still faced the same issues as their predecessors when deploying on a device or they lacked interactions in the AR environment. Our work overcomes the limitations of existing studies by implementing the tool on the web-based environment. In addition, animations of the 3D models and interactions in the AR scene are added.

## 4. Methods

### *4.1. System Design*

This section describes our method and techniques to build BlocklyAR in detail, and readers are encouraged to consult the demonstration video of the proposed tool on YouTube [23].

BlocklyAR was developed using JavaScript libraries and in particular, Blockly [20], A-Frame [11], and AR.js [24]—an open-source library built on top of ARToolkit and integrated with A-Frame.

A-Frame is a web framework for building VR experiences through a custom component. It is built using HTML, which is easy to read, understand, and copy and paste. A-Frame was created to be accessible to everyone, including web developers, enthusiasts, artists, designers, educators, makers, and K-12 students. Nguyen et al. [6,9,25–27] largely confirmed this claim, although some learners still find it difficult to control interactions in the virtual environment. Unlike previous studies that created visual tools for dedicated devices (e.g., Android, iOS), we tried to bring an AR experience to a more general audience with just a smart device (smart phones, tablets with built-in WebGL support).

The primary goal of BlocklyAR is to create a visual programming interface that enables non-programmers (e.g., young learners and enthusiasts) to experience their self-created AR without the effort of memorizing coding syntax. Several considerations were taken into account when designing BlocklyAR, such as (1) expressing programming ideas without memorizing syntax of coding, (2) assessing the AR application from the expressions promptly, (3) generating the programmed AR application with minimal effort, and (4) sharing the newly created AR application with others. To meet these goals, we followed the visual design suggestion by Munzner [28] in which the requirements of the application were classified into tasks and the visual design was carried out to fulfill these tasks. The proposed BlocklyAR tool has the following high-level tasks:

- Task 1 (T1). It enables users to program an AR application visually.
- Task 2 (T2). It supports multiple content types, including text, sound, primitive models, and external 3D models (e.g., GLTF, OBJ).
- Task 3 (T3). It allows young learners to generate both AR content and animations associated with it.
- Task 4 (T4). It enables users to experience their coding scheme on the visual interface.
- Task 5 (T5). It allows users to share AR applications they have developed with others.

Based on the tasks outlined above, BlocklyAR was designed with the following three primary components: (1) the coding editor, (2) the visual AR, and (3) the supporting panel. The following subsection describes each component in detail.

#### 4.1.1. The Coding Editor

As depicted in Figure 2, the Blockly coding editor of BlocklyAR consists of two parts: a toolbox and a workspace. The toolbox works as an entry point where users can select the appropriate visual cues, and then drag and drop them into the workspace (Task T1). By default, the workspace includes a trashcan that enables users to remove unwanted blocks.

The main value of Blockly is the inclusion of customizable blocks along with some existing blocks for common operations. Due to this unique feature, Blockly can generate code in any textual programming language. However, new block types each require a definition and a generator. The definition describes the block's appearance and how it behaves, including the text, color, shape, and the connections to other blocks while the generator translates the block to executable code.

Our tool uses an A-Frame library for rendering virtual objects. Thus the executable code is attributed to A-Frame's TypeScript. The architecture of A-Frame is built up on the entity–component–system (ECS). ECS provides developers flexibility for defining objects by reusing existing parts, which makes designing cleaner. In ECS, an entity is defined as an empty object container where a component or a set of components can be attached. Without any component, an entity does not render anything. Component is a reusable module or data container that provides appearance, behavior, or functionality. All logic of an object is implemented through a component. Thus, by mixing different components, a new type of object can be defined. Global scope, management, and services of components are managed through the system.

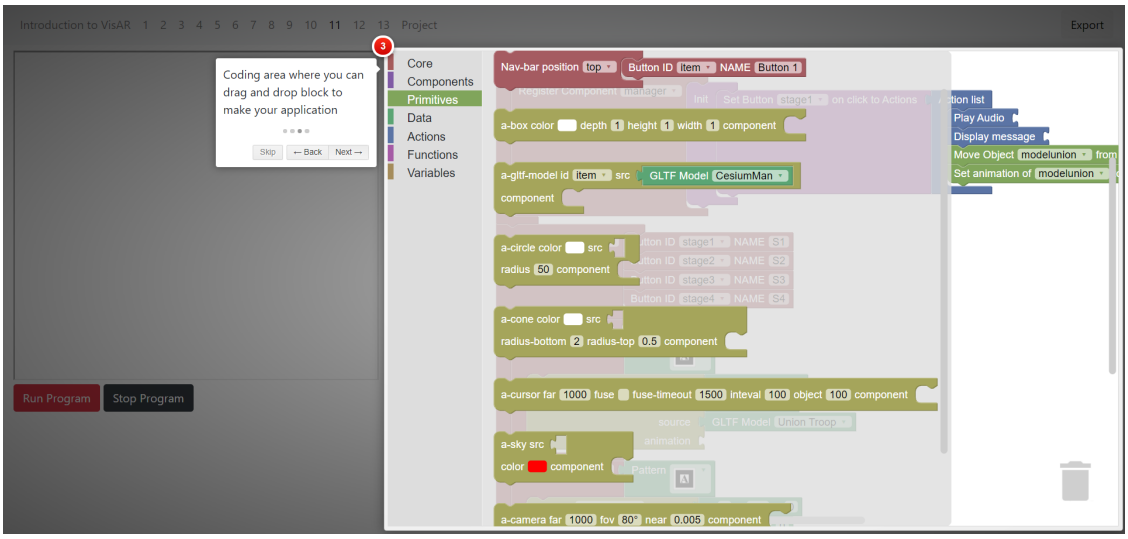


**Figure 2.** The coding editor of BlocklyAR enables users to drag and drop a palette of commands into the working space.

For initiating the camera, and detecting and tracking objects' positions, the AR.js library was used as a component plugin for A-Frame. Initially, A-Frame was designed only for rendering VR, so predefined components are mostly dedicated to the VR environment. On the other hand, AR.js allows developers to detect images and patterns, estimate spatial positions, and superimpose virtual objects on the detected images/patterns. AR.js does not provide any mechanism to control the behavior of virtual objects. Thus we have to manually define some behaviors for a particular object and model. In our work, we extract only components that can be used in conjunction with AR.js and we define blocks for generating these components. Figure 2 shows our defined blocks, which are grouped into seven categories, including core, components, primitives, data, actions, functions, and variables. Each category is represented by a unique color for differentiating among them.

We describe the technical details for the translation from block to coding so that enthusiasts and motivated students can duplicate or replicate our work as follows:

1. Core: There are four block types in these category consisting of AR scene, marker, entity, and script. The AR scene block is translated into a typescript tag as "<a-scene arjs></a-scene>". Similarly, marker and entity blocks are transcribed as "<a-marker>...</a-marker>" and "<a-entity>...</a-entity>" respectively. The Script block is a special cue that allows creators to inject arbitrary JavaScript into the AR app. The script tag is often inserted at the end of the HTML document body. However, in A-Frame components should be declared first so the Script block must be placed before the AR scene block. The Script block is translated as "<script>...</script>". The three dots "..". in the translated tags indicate that we can nest other entities within tags.
2. Components: Currently, A-Frame provides 42 components for users to work with. As mentioned earlier, most components are used for navigation and interactions in the VR environment (hand control, look control, keyboard control or Oculus/HTC Vive control, etc.). Thus, we only keep 10 general components, as follows: position, rotation, scale, camera, look-at, background, geometry, material, visible, and light. The shape of the component blocks is shown in Figure 3; the default value for each block is adapted from the A-Frame components. Each block and its translation are described in detail as follows:

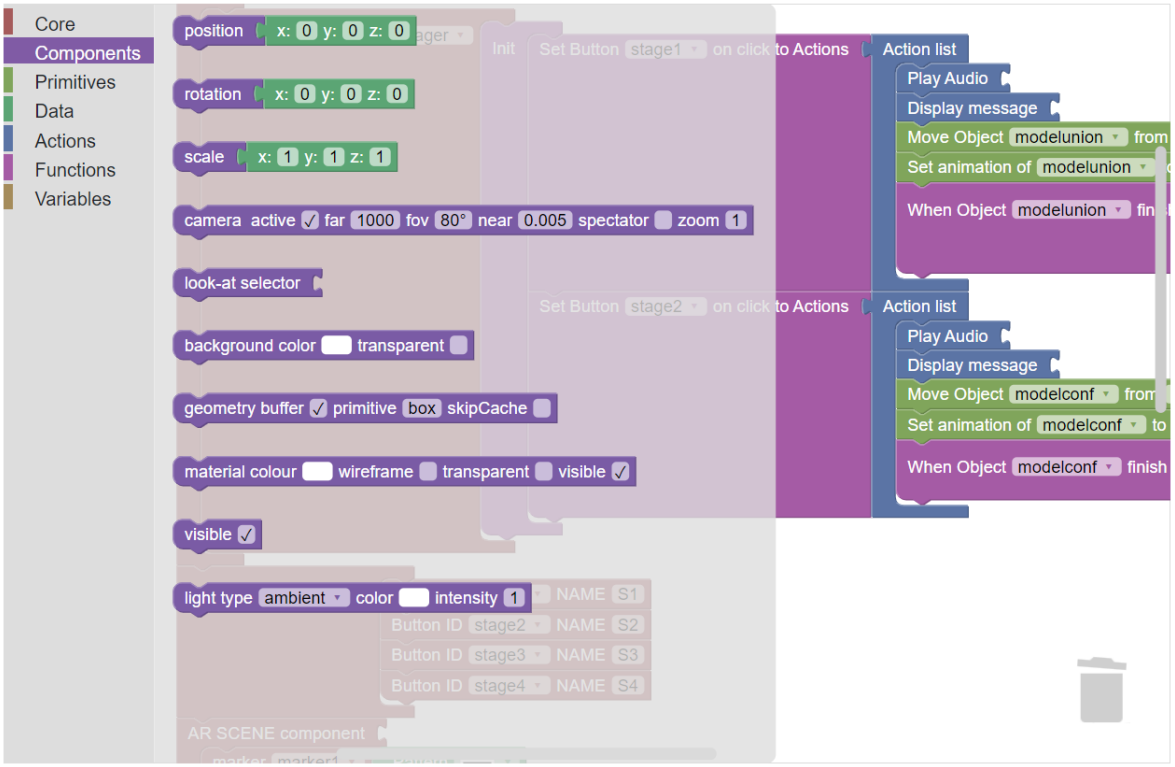


**Figure 3.** The coding editor of BlocklyAR enables users to drag and drop a palette of commands into the working space.

- Position: This component places an entity at certain location in 3D space. It takes a coordinate value from three space-delimited numbers (i.e., "0 1 2" corresponds to x, y, and z position accordingly). In the AR app, the position of an object is relative to its marker. The position block is translated as "position = value" where value is the input from Vector3 block (which is described in the Data category). If the position component is not explicitly attached to an entity, the entity is assumed to be located at the origin (position = "0 0 0") in the 3D space or at the center of the marker.
- Rotation: This component sets the orientation of an entity in degrees. It takes the pitch (x), yaw (y), and roll (z) to indicate degrees of rotation. The rotation block is translated as "rotation = value" where value is the input from Vector3 block.
- Scale: The scale component defines the transformation of an entity such as shrinking, stretching, or skewing along the axes x, y, and z. respectively. The scale block is translated as "scale = value" where value is the input from Vector3 block.
- Camera: The camera component identifies from which perspective the user views the scene. Properties of the camera component include active (whether this camera is active), far (camera frustum far clipping plane), fov (field of view), near (camera frustum near clipping plane), spectator (used to render a third-person view), and zoom (zoom factor of the camera). The translated code for this component is "camera ="active: input_value; far: input_value; fov: input_value; near: input_value; spectator: input_value; zoom: input_value"" where input_value is defined by users from the block.
- Look-at: This component defines the orientation behavior of an entity over time. It updates the rotation component in such a way that the entity always faces toward another entity or position. The translated code for this component is "look-at ='value'" where value can be either a Vector3 block (in the Data category) or an identifier of an object (in the Variables section).

- Background: The background component defines a basic color background of a scene. It takes two inputs as color and transparency. The background block is translated as "background = 'color: input_value; transparent: boolean'" where the input_value and boolean are set by users.
- Geometry: This is a universal component to define the shape of an entity. The input of this component can be either data defined by users or primitive shape (which is described in the Primitives category). The translated code from the geometry block is "geometry ='buffer: boolean; primitive: input_value; skipCache: boolean'".
- Material: The material component defines the appearance as an entity such as color, wireframe, transparency, or visibility. The translated code from the geometry block is "material ='color: input_value; wireframe: boolean; transparent: boolean; visible: boolean'".
- Visible: This component determines whether to render an object in the scene or not. The translated code from the visible block is "visible = boolean". If the boolean value is false, the object is not drawn and vice versa.
- Light: This component defines the entity as a source of light. Its properties include type of light (ambient, directional, hemisphere, point, spot), color of the light, and the intensity (light strength). We translate the light block into the component as "light = 'type: input_value; color: input_value; intensity: input_value'".

3. Primitives: BlocklyAR supports 27 visual cues that can be translated into primitives available in A-Frame. In addition, our tool helps to generate HTML tags such as div or button. The mechanism to generate code for the primitives is similar to entity block. Due to a large number of primitives, we do not describe all of them in detail in this section. Instead, we describe blocks that are often used in our AR application.

   - Box: This primitive geometry enables the system to draw a box in the scene. Basically, it is defined by color, height, width, and depth. The box is put in BlocklyAR for testing purposes. The translation from block to A-Frame tag as "<a-box color='input_value' depth='input_value' width='input_value' height='input_value'>...</a-box>". The ellipses mean that other boxes or entities can be nested in this box.
   - gltf: This is one of the most important blocks that makes BlocklyAR unique (Task 2). Its translation enables the system to render a 3D object and animations in GLTF format [29]. We use these animations to make the characters in the scene perform behaviors. The available animations were extracted in the animation lists block (described in the Data category). The translation from this block is defined as "<a-gltf-model id="input_variable" src="input_value"></a-gltf-model>" where input_variable is retrieved from block in the Variables category and input_value comes from Data category.

4. Data: As its name implies, this block allows users to define the data type, default value, and constraint of a given block. Blocks in this category are usually used as inputs for the component and primitive blocks. We define seven block types dedicated for the AR application.

   - Vector3: This block defines a coordinate value as three space-delimited numbers. The translation of this block is "x y z" corresponding to x, y, and z axes in the 3D space. The output of this block is used to set the position, location, and scale of an object.
   - Animation lists: This block defines all available animations of GLTF models (Task 3). Its output gives instruction to the system for playing a particular animation of a 3D object. The translation for this block is "animation_type" where animation_type can be idle, walking, running, etc.
   - Audio lists: Similar to animation list block, the audio lists blocks determines a set of available audio files in the system (Task 2). Its output provides the audio location as "audio_path".

- Preset: The preset block defines the type for the marker block. It is translated as "preset = ’dropdown_value’" where dropdown_value includes hiro and kanji - two popular markers for testing AR application.
- Pattern: Users can use custom markers for their AR application. This block determines the marker and its pattern location and is translated as "pattern = ’pattern_path’" where pattern_path is the location of the pattern file.
- GLTF: This block specifies the location of the GLTF model (Task 2). It is used as a source for the gltf block in the Primitives category.
- Text: This block defines the text content to be displayed or used by other blocks. Its output is "text_content" (Task 2).

5. Actions: This category consists of blocks that define typical atomic events in the AR application. The shape of each of the blocks is illustrated in Figure 4.

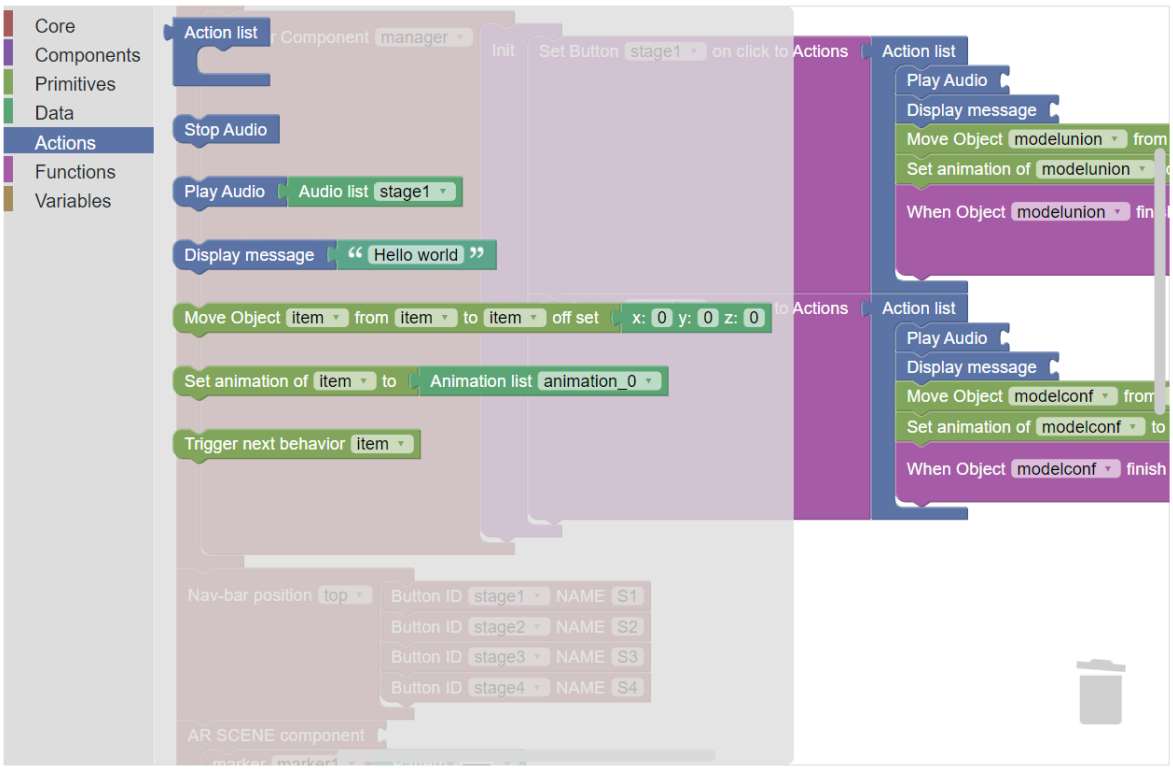


**Figure 4.** The coding editor of BlocklyAR enables users to drag and drop a palette of commands into the working space.

- Action list: This block defines an empty action container that allows a series of event to be performed within it. The translation of this block is an anonymous function in JavaScript as "()=>{...}" where "..". represents the atomic event.
- Stop audio: This block definition enables users to stop playing all audio in the AR app. It traverses the HTML DOM (Document Object Model) elements, finds the audio objects, and triggers the pause event. The block is translated into JavaScript command as "$(’audio’).trigger(’pause’)".
- Play audio: This block is translated into an executable code where a particular audio object is triggered. It is transcribed as "$(’#input_value’).trigger(]play])", where #input_value is the identifier of the audio object.

- Display message: The display message block defines a message to be displayed on the screen (Task 2). It takes the text block as the input. The literal translation of this block is "$('body').appendChild('input_value')".
- Move object: This is one of the most important blocks in BlocklyAR since it allows a particular 3D model to travel and move between markers (Task 3). This literal block identifies which 3D object involves movement from which location and where the destination is. Due to a large number of command lines, we do not present the translation here. Scholars can find it on our GitHub repository.
- Set animation: When the set animation block is used, it prompts the system to play a specified animation of a particular 3D object (Task 3). The translated code for this block is "$('#object_identifier').attr('animation-mixer','clip: animation_type')"
- Trigger next action: This block defines a transition between a different stage of an application or simulates an event. Particularly, it takes an input, an HTML element and triggers the "click" event. The translation is described as "$('#object_identifier').trigger('click')"

6. Functions: The category contains blocks that define the sequence of events or the object's behaviors over time. The arrangement of blocks allow ordering of the execution codes.

   - Set actions to an object: This block enables users to create a series of actions for a particular object. These actions are taken from the Action List block. The translation is described as "$('#object_identifier').on('click', action_list)".
   - Next actions: This block is used when users want to perform other actions when a given object finishes running actions. The next action block is translated as "$('#object_identifier').on('end', action_list)" where action_list contains the next instructions (i.e., the Action_List block).
   - Wait: This block defines a delay time for the next execution. Its translation is expressed as "setTimeout(function(){actions},second})" where actions are taken from the action blocks and second is the delay time.

7. Variables: This category enables users to define variables in the AR application. Since BlocklyAR allows the addition of any arbitrary 3D objects in the scene, each object needs to have a unique identifier. The variable block defines the variable name to be used in other blocks. The variable block is provided by the Blockly library.

#### 4.1.2. The Visual Augmented Reality Component

Similar to stage area in the Scratch environment where users are able to see the results of the program execution, the visual AR component enables enthusiasts to experience their coding schemes in the mixed 3D space (Task 4). This component is wrapped by a sandbox (i.e., iframe) to separate the BlocklyAR coding mechanism from the AR execution programs since both applications use JavaScript programming language. This visual component defines an AR template which imports two libraries (e.g., A-Frame and AR.js) and contains no content. Our initial approach for rendering AR contents is to use hot reloading, meaning that every change in the coding editor component will be reflected in the AR component immediately, notwithstanding the trade-off for maintaining the states between the two components. As an alternative we define a "Run Program" button that is available for programmers to initiate the AR scene and run the program executions. Users can also stop the AR application with "Stop program" functionality while focusing on scripting. Figure 5 illustrates an example extracted from our use case when users want to add two different 3D characters along with their poses (i.e., idle and rest) on top of the markers (i.e., letter A and B) and then execute the program instructions or "Run Program" (Task 4).

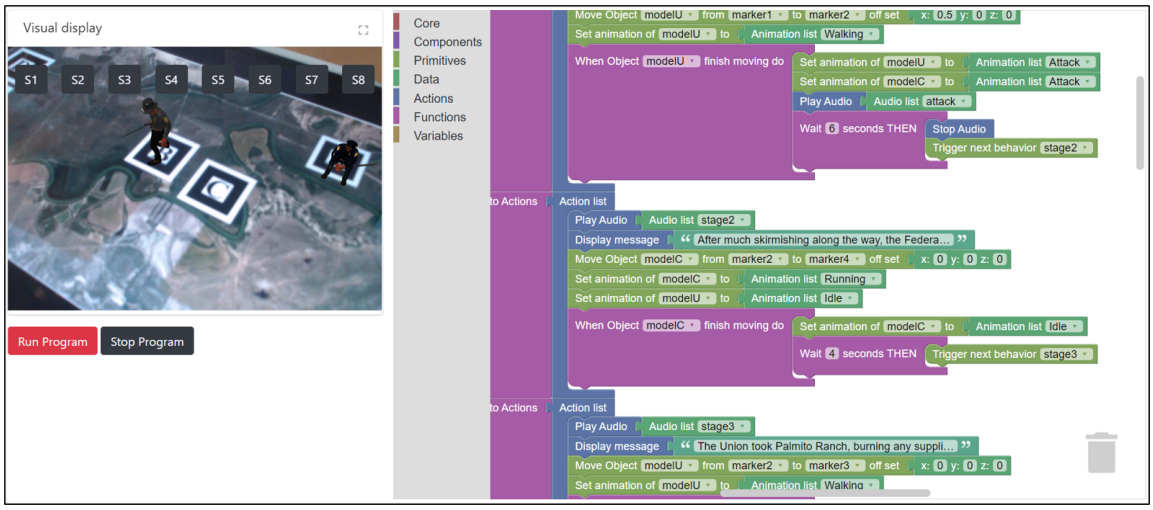


**Figure 5.** The visual AR component enables enthusiasts to experience their coding schemes in the mixed 3D space.

#### 4.1.3. The Supporting Panel Component

While the other two components assume that users have a basic knowledge of using BlocklyAR, this component provides an introduction to our tool as well as tutorials to generate an AR application step by step, beginning with a simple task and moving to a more complex one. The "Introduction" section enables creators to get an overview layout of BlocklyAR where each component is highlighted and supported by a description. Thirteen tutorials are provided to guide learners on how to use blocks and the connections among them. Each tutorial was constructed based on the previous one and accompanied by a description (as depicted in Figure 6). The supporting panel also includes two options that enable users to load existing workspace as well as export the AR application for sharing in HTML format (Task 5).

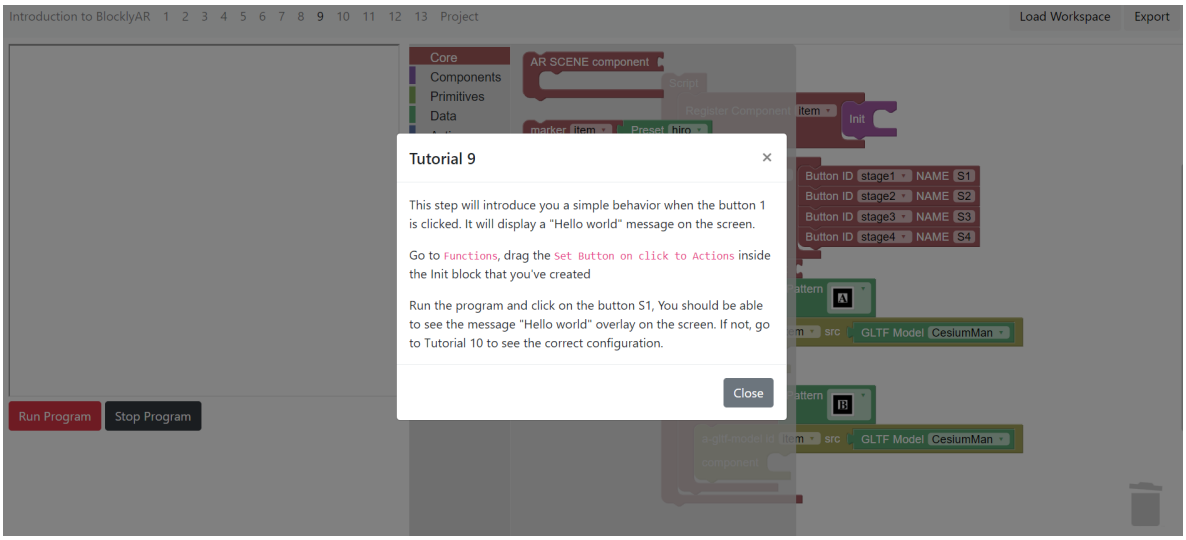


**Figure 6.** Tutorial section where learners are guided on how to use blocks and the connections among them.

### *4.2. Use Case*

This section describes a use case where enthusiasts are able to reproduce existing work by using BlocklyAR. The context of this study was retrieved from the work of Nguyen et al. [3,30] where the authors reenacted the historic battle of Palmito Ranch in Brazos Santiago, Texas that occurred on 12–13 May 1865. This work also used A-Frame and AR.js to create a 3D environment and superimpose

virtual objects on the markers, respectively. This work was selected for our study because of their AR-application-engaged animations and moving of 3D objects in the scene. We did not bring other work into consideration if the task simply showed the information. Although the AR application (PalmitoAR) can fulfill the tasks the authors laid out, it was written in pure JavaScript and thus required a huge amount of work to control every action and animation in the scene. The PalmitoAR defined eight stages to be simulated and we transcribed four stages into the action blocks as follows (from stage 5 to stage 8 are almost identical to previous ones):

- Stage 1 (S1): The Union began moving from White Ranch toward Palmito to attack the Confederacy—play audio, display message, move the Union 3d models from a marker to another marker, and set animation of the Union models to "walking". When the Union finishes moving set animation of the Union and Confederacy to "attack", play audio "attack", wait 6 s, and then transition to the next stage (S2).
- Stage 2 (S2): The Confederacy scattered to different locations—play audio, display message, move the Confederate 3d models from a marker to another marker, set animation of the Union to "idle", and set animation of the Confederacy to "running". When the Confederacy finishes the movements, set animation of the Union to "idle", wait for 6 s, and then transition to next the stage (S3).
- Stage 3 (S3): The Union burned supplies and remained at the site to feed themselves and their horses—play audio, display message, move the Union from a marker to another marker, set animation of the Confederacy to "walking". When the Union finishes the movements, set animation of the Union to "resting", wait for 6 s, and then transition to the next stage (S4).
- Stage 4 (S4): The Confederacy arrived with reinforcements and forced the Union back to White Ranch—play audio, display message, move 3D models from a marker to another maker, and set animation of all models to "walking". When the 3D models finish their movements, set animation of all models to "attack". Wait for 6 s and then move the Union from a marker to another marker, set animation of the Union to "running", and then set animation of the Confederate to "idle". When the Union finishes the movements, set animation of the Union to "idle", wait for 6 s, and then transition to next the stage (S5).

Based on the perceptual conversation above, we can drag and drop the corresponding blocks to their appropriate positions. Figure 7 illustrates the set of block instructions. The data for models, markers, audio, or a text message was cloned from PalmitoAR's GitHub repository. When running the AR application based on the instruction we created, our result could yield the same quality compared to the original Palmito's version. The demonstration of this use case is available on YouTube [23].

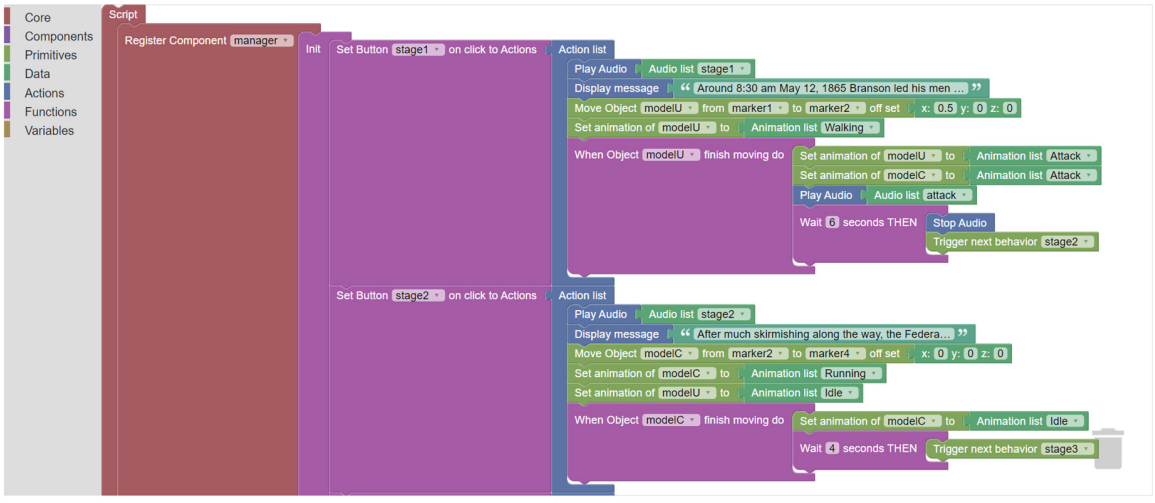


**Figure 7.** Use case of using BlocklyAR to recreate the Palmito Battle Ranch [3].

### *4.3. Evaluation*

To evaluate the BlocklyAR, we adapted the technology acceptance model (TAM) originally proposed by Davis [31]. The TAM model highlighted the importance of users' two main perceptions affecting technology adoption behavior, i.e., perceived usefulness and perceived ease-of-use. Perceived usefulness refers to "the degree to which a person believes that using a particular system would enhance his or her job performance", while perceived ease-of-use is defined as "the degree to which an individual believes that using a particular system would be free from effort". Davis et al. [32] showed that perceived usefulness positively influences users' intentions to use technology, whereas perceived ease-of-use is secondary, mediated by perceived usefulness. The TAM model has been widely used in various domains with different variations. Dishaw and Strong [33] incorporated task–technology fit to the existing TAM model to investigate whether the task–technology fit will influence the performance outcome of an application. Task technology fit refers to "the degree to which a technology assists an individual in performing his or her portfolio of tasks". They asserted that the proposed technology must be utilized and fitted with the tasks it supports in order to achieve good impacts on individual performance. Their findings suggested that task–technology fit positively influences perceived ease-of-use. Visual design (or visually appealing) is another variable used to measure whether the layout will impact the customers' retention through increasing trust and loyalty [34]. Verhagen et al. [35] showed that this variable positively influences perceived usefulness. In our study, we extended the TAM model with two external variables, task–technology fit and visual design.

### *4.4. Research Model*

The extended TAM model examines the mediating role of perceived usefulness and perceived ease-of-use in their relationship between the external variables (task–technology fit and visual design) and the probability of technology use (intention to use). Specifically, we evaluate the following hypotheses for users' acceptance of BlocklyAR: perceived visual design positively influences perceived task–technology fit (H1), perceived task–technology fit positively influences perceived ease-of-use (H2), Perceived visual design positively influences perceived usefulness (H3), perceived ease-of-use positively influences perceived usefulness (H4), perceived ease-of-use positively influences intention to use (H5), and perceived usefulness positively influences intention to use (H6). The proposed hypotheses are transcribed into the research model as illustrated in Figure 8, where each set of ellipses represents a construct (also called a latent variable measured by a set of items) and arrows denote the hypotheses from 1 to 6.

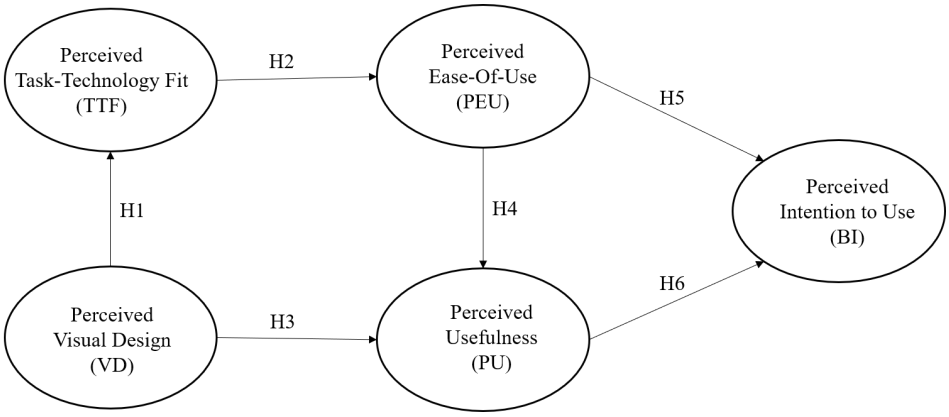


**Figure 8.** The conceptual research model adapted from [31] with extensions of task–technology fit and visual design variables. Each set of ellipses represents a construct and an arrow denotes a hypothesis.

### *4.5. Data Collection and Analysis*

Data for analysis was collected from the two main sources: (1) at the site of the study, and (2) from Amazon Mechanical Turk (AMT). Due to the COVID-19 pandemic, we were unable to collect sufficient suitable subjects from the first source (only 17 participants were collected at the site), which necessitated to the use of the second source as a supplementation. Google Form was used as an online instrument to collect data. Before the survey was administered to participants, our study was approved by the University's Institutional Review Board to recruit participants to take part in the survey. Each participant was compensated for their voluntary time and no sensitive personal information was collected. The survey consists of three parts: (a) two YouTube videos participants were required to watch, (b) 20 Likert scale questions for opinions and one open-ended question for comments/suggestions, and (c) four questions for general demographic information. In the first part, participants were asked to watch a demo application of AR [36] and our proposed toolkit [23]. After participants finished watching the videos, they were asked to respond to questions about their attitudes, comments, and behavioral intention to use BlocklyAR with 5-point Likert scales (which ranged from "strongly disagree(1)" to "strongly agree(5)") and one open-ended question. Table 1 shows the set of items (or survey questions) used to measure each construct. The final part asked subjects to provide general information such as gender, English as a first language, and ethnicity. The survey was administered to 24 subjects at the site of study and 75 subjects on Amazon Mechanical Turk. We received 18 responses from on site and 75 from AMT. Data were cleaned by removing irrelevant responses such as incomplete or fake responses (also known as using automated tool to answer the survey). In total, 66 subjects (17 on site, and 49 on AMT) participated in the study. Table 2 shows a summary of participants' demographic information: 80.3% of the participants were male and 18.18% were female with ages ranging from 23 to 57 (mean = 34.3). More than half the participants reported English as a first language (59.09%) and 40.91% did not. A majority of the participants was Asian (59.09%), 27.27% were Caucasian, 12.82% were American Indian/Alaska Native, 4.55% were Hispanic, and 1.52% were African American/Black.

**Table 1.** Construct and items.

| Construct | Source |
|---|---|
| **Perceived task–technology fit**<br>(TTF1) The application is adequate for AR-based visual programming education.<br>(TTF2) The application is compatible with the task of controlling virtual objects.<br>(TTF3) The application fits the task (i.e., AR-based visual programming education) well.<br>(TTF4) The application is sufficient for a AR-based visual programming learning toolkit. | [37] |
| **Perceived visual design**<br>(VD1) The visual design of the BlocklyAR is appealing.<br>(VD2) The size of 3D virtual objects is adequate.<br>(VD3) The layout structure is appropriate. | [35] |
| **Perceived usefulness**<br>(PU1) Using BlocklyAR would improve my knowledge in AR-based visual programming skills.<br>(PU2) Using BlocklyAR, I would accomplish tasks (e.g., AR-based visual programming) more quickly.<br>(PU3) Using BlocklyAR would increase my interest in AR-based visual programming.<br>(PU4) Using BlocklyAR would enhance my effectiveness on the task (i.e., AR-based visual programming).<br>(PU5) Using BlocklyAR would make it easier to do my task (i.e., AR-based visual programming). | [38] |
| **Perceived ease of use**<br>(PEU1) Learning to use the AR-based visual programming toolkit would be easy for me.<br>(PEU2) I would find it easy to get the AR-based visual programming toolkit to do what I want it to do.<br>(PEU3) My interaction with the AR-based visual programming toolkit would be clear and understandable.<br>(PEU4) I would find the AR-based visual programming toolkit to be flexible to interact with.<br>(PEU5) It would be easy for me to become skillful at using the AR-based visual programming toolkit.<br>(PEU6) I would find the AR-based visual programming toolkit easy to use. | [38] |
| **Intention to use**<br>(BI1) I intend to use the AR-based visual programming toolkit in the near future.<br>(BI2) I intend to check the availability of the AR-based visual programming toolkit in the near future. | [38] |

**Table 2.** General information about the participants.

| Variable | Category | Number | Percentage |
|---|---|---|---|
| Gender | Male | 53 | 80.30 |
| | Female | 12 | 18.18 |
| | Not to say | 1 | 1.52 |
| English as a First Language | Yes | 39 | 59.09 |
| | No | 27 | 40.91 |
| Ethnic heritage | Asian | 39 | 59.09 |
| | Hispanic/Latino | 3 | 4.55 |
| | Caucasian/White | 18 | 27.27 |
| | American Indian/Alaska Native | 5 | 12.82 |
| | African American/Black | 1 | 1.52 |
| Total | | 66 | 100 |

Generalized structured component analysis (GSCA) [39,40] was used for evaluating our proposed research model due to its flexibility to work with a small sample size without requiring rigid normal distribution [41,42]. GSCA is a component-based approach to structural equation modeling, which enables to evaluate the measurement model for the associations between the TAM constructs and their indicators/items (i.e., loadings, reliability), as well as the structural model for the directional relationship among the TAM constructs (i.e., research hypotheses). The web-based tool for GSCA is available online at [43].

## 5. Results

### *5.1. Qualitative Analysis*

Overall, BlocklyAR received positive feedback from participants in the study. For example, subject 59 (S59) commented "I found it really interesting. It [is] easy to design AR related stuff using this tool. It is easy for the non professional coders to use and create AR". S56 indicated that, "This would be very useful for training for kids and starters"; this comment is aligned with our initial intention of making AR accessible to enthusiasts. In terms of introducing the toolkit through video, S30 stated that "The tutorial for using AR is very elaborated with different stages and steps, [which] make it easy to learn and follow. The blocks are [also] easy to grasp". This comment from S9 encouraged us to work harder to improve the toolkit: "This need[s] to be implemented [a] lot in schools, colleges etc. for education, will be very useful for students and learners, for remote teaching. Mainly during situations like recent pandemic due to COVID 19", as did this from S26: "This toolkit is really fantastic and impressive", and this from S10, "this toolkit is easy to understand if you have prior programming knowledge. it is also useful for beginners who don't know programming but want to use AR programming."

Besides the positive comments, participants also gave some feedback for improvements BlocklyAR such as improving the 3D model—"The characters look unfashionable" (S61)—increasing the visibility of the marker—"I will suggest that the size of the lettering be a little bolder" (S50)—and expanding the video introduction—"It would be useful for newbie[s] to approach if there is a part of the video introducing about the main components of how AR applications are built: The camera (for what), markers (for what), and then programming part" (S66) and "It [the video introduction] was a bit fast" (S22).

### *5.2. Quantitative Analysis*

Table 3 shows the descriptive statistics of our proposed constructs. It can be seen that all the means of the TAM measures are above the average point of 3. The standard deviations range from 0.607 to 1.162.

**Table 3.** Means and standard deviations of TAM measures (N = 66).

| Construct | Item | Mean | SD |
|---|---|---|---|
| Perceived Task Technology Fit | TTF1 | 4.409 | 0.607 |
| | TTF2 | 4.364 | 0.694 |
| | TTF3 | 4.439 | 0.659 |
| | TTF4 | 4.091 | 0.799 |
| Perceived Visual Design | VD1 | 4.030 | 0.928 |
| | VD2 | 3.985 | 0.813 |
| | VD3 | 4.212 | 0.755 |
| Perceived Usefulness | PU1 | 4.333 | 0.771 |
| | PU2 | 4.197 | 0.863 |
| | PU3 | 4.258 | 0.933 |
| | PU4 | 4.303 | 0.744 |
| | PU5 | 4.333 | 0.771 |
| Perceived Ease of Use | PEU1 | 3.970 | 0.859 |
| | PEU2 | 3.864 | 0.892 |
| | PEU3 | 4.136 | 0.802 |
| | PEU4 | 4.030 | 0.859 |
| | PEU5 | 4.015 | 0.868 |
| | PEU6 | 4.061 | 0.959 |
| Intention to use | BI1 | 3.636 | 1.145 |
| | BI2 | 3.864 | 1.162 |

Table 4 provides the measures for internal consistency and convergent validity for each TAM construct. Dillon-Goldstein's rho was used to assess the internal consistency reliability criterion for each construct. All the values for each construct are greater than 0.7, thereby exceeding the reliability estimates recommended in [39]. To check convergent validity, we examined the Average Variance Extracted (AVE) value of each latent variable. All AVE values are greater than 0.5, indicating a reasonable convergent validity [39].

**Table 4.** Internal consistency and convergent validity.

| Construct | Item | Dillon-Goldstein's rho | Average Variance Extracted |
|---|---|---|---|
| Perceived task–technology fit (TTF) | 4 | 0.809 | 0.519 |
| Perceived visual design (VD) | 3 | 0.780 | 0.543 |
| Perceived usefulness (PU) | 5 | 0.853 | 0.540 |
| Perceived easy of use (PEU) | 6 | 0.901 | 0.605 |
| Intention to use (BI) | 2 | 0.929 | 0.868 |

Table 5 presents the loading estimates for the items along with their standard errors (SEs) and 95% bootstrap percentile confidence intervals (CIs) with the lower bounds (LB) and the upper bounds (UB), calculated from 100 bootstrap samples. Here, a parameter estimate is assumed to be statistically significant at 0.05 alpha level if the CI does not include the value of zero. All the loading estimates were statistically significant, indicating that all those items were good indicators of the constructs.

Table 6 provides the estimates of the directional path coefficients (i.e., the research hypotheses) in the structural model along with their standard errors and 95% confidence intervals, which were obtained from GSCA analysis with 100 bootstrap samples by fitting the hypothesized technology acceptance model to the dataset. Results showed that visual design had statistically significant and positive influences on task–technology fit (H1 = 0.439, SE = 0.124, 95% CI = 0.175–0.686). Task–technology fit had a statistically significant and positive influence on perceived ease of use (H2 = 0.434, SE = 0.118, 95% CI = 0.184–0.678). Moreover, perceived usefulness had statistically significant and positive effects on intention to use (H6 = 0.319, SE = 0.136, 95% CI = 0.026–0.574), and perceived ease-of-use had statistically significant and positive effects on intention to use (H5 = 0.417, SE = 0.134, 95% CI = 0.132–0.662). However, hypotheses H3 (visual design → perceived usefulness) and H4 (perceived ease of use → perceived usefulness) were not supported due to the inclusion of zero values in CIs.

**Table 5.** Estimates of loadings.

| | Estimate | SE | 95%CI_LB | 95%CI_UB |
|---|---|---|---|---|
| TTF1 | 0.809 | 0.046 | 0.712 | 0.892 |
| TTF2 | 0.544 | 0.099 | 0.270 | 0.680 |
| TTF3 | 0.787 | 0.086 | 0.586 | 0.892 |
| TTF4 | 0.711 | 0.103 | 0.428 | 0.844 |
| VD1 | 0.677 | 0.129 | 0.293 | 0.852 |
| VD2 | 0.787 | 0.097 | 0.572 | 0.887 |
| VD3 | 0.743 | 0.142 | 0.326 | 0.857 |
| PU1 | 0.762 | 0.068 | 0.612 | 0.871 |
| PU2 | 0.802 | 0.053 | 0.698 | 0.904 |
| PU3 | 0.732 | 0.062 | 0.574 | 0.829 |
| PU4 | 0.724 | 0.056 | 0.592 | 0.816 |
| PU5 | 0.644 | 0.087 | 0.465 | 0.810 |
| PEU1 | 0.765 | 0.054 | 0.648 | 0.862 |
| PEU2 | 0.770 | 0.067 | 0.598 | 0.872 |
| PEU3 | 0.762 | 0.081 | 0.570 | 0.866 |
| PEU4 | 0.726 | 0.050 | 0.631 | 0.817 |
| PEU5 | 0.809 | 0.045 | 0.709 | 0.884 |
| PEU6 | 0.829 | 0.037 | 0.755 | 0.888 |
| BI1 | 0.930 | 0.025 | 0.877 | 0.971 |
| BI2 | 0.933 | 0.026 | 0.870 | 0.971 |

**Table 6.** Estimates of path coefficients.

| | Estimates | Std.Error | 95%CI_LB | 95%CI_UB |
|---|---|---|---|---|
| VD → TTF | 0.439 * | 0.124 | 0.175 | 0.686 |
| VD → PU | 0.127 | 0.208 | −0.278 | 0.464 |
| TTF → PEU | 0.434 * | 0.118 | 0.184 | 0.678 |
| PEU → PU | 0.212 | 0.161 | −0.066 | 0.554 |
| PU → BI | 0.319 * | 0.136 | 0.026 | 0.574 |
| PEU → BI | 0.417 * | 0.134 | 0.132 | 0.662 |

* statistically significant at 0.05 level.

## 6. Discussion

Our study has several limitations that should be addressed in the future research. The first limitation is the procedure to collect user responses. This was due to the COVID-19 pandemic that prevents us from conducting the study in a face-to-face fashion. In addition, by only watching the video, participants were unable to use the toolkit directly, which may have reduced the motivation to take part in the survey. As such, more rigorous research would be needed to evaluate the adaption and use of BlocklyAR, even though it is not uncommon to collect user responses by watching videos [44–47]. Second, BlocklyAR did not support an arbitrary action or actions defined by users; we acknowledge that the action space is huge and users may be interested in only a certain action depending on a given domain. In fact, BlocklyAR can be considered as an abstract or a high-level programming interface for A-Frame combined with AR.js, so we only defined elements that are most commonly be used in an AR application with an extension of controlling the animations and movement of a 3D object. Enthusiasts can refer to the technical detail in Section 4.1 for replicating and extending the work. Third, the current version of BlocklyAR only supports the marker-based approach, meaning that users have to prepare a marker and put it in front of the camera. We have not taken advantage of WebXR yet, due to the unavailability of a stable WebXR API as well as our lack of compatible devices for conducting the experiment. Fourth, privacy concerns were not taken into account in this study. Rauschnabel et al. [48] discussed that information captured by a device's sensors might threaten the privacy of both users and other people, thereby placing an obstacle for using AR technology. Lastly, other factors contributing to the adoption and use of technology might be considered in future studies, which have been discussed in the unified theory of acceptance and use of technology (UTAUT) and its extensions [49,50]. That is, it would extend the technology adoption framework used in this study by evaluating the influences of

performance expectancy, effort expectancy, social influence, facilitating conditions, hedonic motivation, price value, and habit on the adoption and use of BlocklyAR, as well as the moderating effects of individual differences (age, gender, and experience) on the constructs

## 7. Conclusions

This paper introduced BlocklyAR, a novel web-based visual programming interface for creating and generating an augmented reality application. By integrating A-Frame and AR.js toolkit into Blockly, BlocklyAR enables young learners and enthusiasts to create AR experiences. The proposed toolkit can be generalized and extended to many other domains for the use of pedagogical and instructional design with animated 3D models, such as demonstrating the fundamentals of electric circuits, testing/simulating robot movements, or assembling hardware components. Following this approach, users can download free 3D models on the internet [51] and then apply animations on them by an intermediate tool presented in [3]. We demonstrated BlocklyAR with a use case where the toolkit can replicate existing work with fewer efforts in programming. Data collected from users' responses indicated that BlocklyAR was useful in learning and making an AR application, with particular relevance for new learners. We used the technology acceptance model to assess users' behavior toward using the toolkit in terms of visual design, task–technology fit, perceived usefulness, perceived ease-of-use, and intention to use. Our findings showed that visual design had statistically significant and positive influences on task–technology fit. Task technology fit had a statistically significant and positive influence on perceived ease-of-use, perceived usefulness and had statistically significant and positive effects on intention to use; and perceived ease-of-use had statistically significant and positive effects on intention to use. However, hypotheses H5 (visual design → perceived usefulness) and H1 (perceived ease-of-use → perceived usefulness) were rejected. Future work will be focusing on replacing the printed map with a virtual real world map (e.g., Mapbox, OpenStreetMap) to increase the fidelity of the AR scene. In this regard, real world location such as latitude and longitude will be used as a substitution for markers.

**Author Contributions:** Conceptualization, V.T.N. and K.J.; data curation, V.T.N. and K.J.; formal analysis, K.J. and V.T.N.; funding acquisition, K.J.; investigation, V.T.N. and K.J.; methodology, V.T.N. and K.J.; project administration, K.J.; resources, V.T.N. and K.J.; software, V.T.N. and K.J.; supervision, T.D. and K.J.; validation, V.T.N. and K.J.; visualization, K.J. and V.T.N.; writing—original draft, V.T.N. and K.J.; writing—review and editing, T.D., V.T.N. and K.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was partially supported by Texas Tech College of Education Competitive Edge Grant.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Azuma, R.T. A survey of augmented reality. *Presence Teleoperators Virtual Environ.* **1997**, *6*, 355–385. [CrossRef]
2. Bruno, F.; Barbieri, L.; Marino, E.; Muzzupappa, M.; D'Oriano, L.; Colacino, B. An augmented reality tool to detect and annotate design variations in an Industry 4.0 approach. *Int. J. Adv. Manuf. Technol.* **2019**, *105*, 875–887. [CrossRef]
3. Jung, K.; Nguyen, V.T.; Yoo, S.C.; Kim, S.; Park, S.; Currie, M. PalmitoAR: The Last Battle of the US Civil War Reenacted Using Augmented Reality. *Int. J. Geo-Inf.* **2020**, *9*, 75. [CrossRef]
4. Norouzi, N.; Bruder, G.; Belna, B.; Mutter, S.; Turgut, D.; Welch, G. A systematic review of the convergence of augmented reality, intelligent virtual agents, and the internet of things. In *Artificial Intelligence in IoT*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 1–24.
5. Azuma, R.; Baillot, Y.; Behringer, R.; Feiner, S.; Julier, S.; MacIntyre, B. Recent advances in augmented reality. *IEEE Comput. Graph. Appl.* **2001**, *21*, 34–47. [CrossRef]
6. T. Nguyen, V.; Hite, R.; Dang, T. Web-Based Virtual Reality Development in Classroom: From Learner's Perspectives. In Proceedings of the 2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), Taichung, Taiwan, 10–12 December 2018; pp. 11–18. [CrossRef]

7. Linowes, J.; Babilinski, K. *Augmented Reality for Developers: Build Practical Augmented Reality Applications with Unity, ARCore, ARKit, and Vuforia*; Packt Publishing Ltd.: Birmingham, UK, 2017.
8. Kato, H. ARToolKit: Library for Vision-Based augmented reality. *IEICE PRMU* **2002**, *6*, 2.
9. Nguyen, V.T.; Jung, K.; Dang, T. Creating Virtual Reality and Augmented Reality Development in Classroom: Is it a Hype? In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), San Diego, CA, USA, 9–11 December 2019; pp. 212–2125.
10. Danchilla, B. Three.js framework. In *Beginning WebGL for HTML5*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 173–203.
11. Mozilla. A Web Framework for Building Virtual Reality Experiences. 2019. Available online: https://aframe.io (accessed on 23 January 2020).
12. WebAssembly. World Wide Web Consortium. 2020. Available online: https://webassembly.org/ (accessed on 5 April 2020).
13. Weintrop, D. Block-based programming in computer science education. *Commun. ACM* **2019**, *62*, 22–25. [CrossRef]
14. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for all. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
15. Radu, I.; MacIntyre, B. Augmented-reality scratch: A children's authoring environment for augmented-reality experiences. In Proceedings of the 8th International Conference on Interaction Design and Children, Como, Italy, 3–5 June 2009; pp. 210–213.
16. CoSpaces. Make AR & VR in the Classroom. 2020. Available online: https://cospaces.io/edu/ (accessed on 5 April 2020).
17. Laine, T.H. Mobile educational augmented reality games: A systematic literature review and two case studies. *Computers* **2018**, *7*, 19. [CrossRef]
18. Wu, H.K.; Lee, S.W.Y.; Chang, H.Y.; Liang, J.C. Current status, opportunities and challenges of augmented reality in education. *Comput. Educ.* **2013**, *62*, 41–49. [CrossRef]
19. Klopfer, E.; Squire, K. Environmental Detectives—The development of an augmented reality platform for environmental simulations. *Educ. Technol. Res. Dev.* **2008**, *56*, 203–228. [CrossRef]
20. Inc, G. Blockly: A JavaScript Library for Building Visual Programming Editors. 2020. Available online: https://developers.google.com/blockly (accessed on 5 April 2020).
21. Mota, J.M.; Ruiz-Rube, I.; Dodero, J.M.; Arnedillo-Sánchez, I. Augmented reality mobile app development for all. *Comput. Electr. Eng.* **2018**, *65*, 250–260. [CrossRef]
22. Clarke, N.I. Through the Screen and into the World: Augmented Reality Components with MIT App Inventor. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2019.
23. Anonymous. BlocklyAR: A Visual Programming Interface for Creating Augmented Reality Experience. 2020. Available online: https://youtu.be/lSsQd8GTcQ8 (accessed on 1 June 2020).
24. Etienne, J. Creating Augmented Reality with AR.js and A-Frame. 2019. Available online: https://aframe.io/blog/arjs (accessed on 23 January 2020).
25. Nguyen, V.T.; Hite, R.; Dang, T. Learners' Technological Acceptance of VR Content Development: A Sequential 3-Part Use Case Study of Diverse Post-Secondary Students. *Int. J. Semant. Comput.* **2019**, *13*, 343–366. [CrossRef]
26. Nguyen, V.T.; Zhang, Y.; Jung, K.; Xing, W.; Dang, T. VRASP: A Virtual Reality Environment for Learning Answer Set Programming. In Proceedings of the International Symposium on Practical Aspects of Declarative Languages, New Orleans, LA, USA, 20–21 January 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 82–91.
27. Jung, K.; Nguyen, V.T.; Diana, P.; Seung-Chul, Y. Meet the Virtual Jeju Dol Harubang—The Mixed VR/AR Application for Cultural Immersion in Korea's Main Heritage. *Int. J. Geo-Inf.* **2020**, *9*, 367. [CrossRef]
28. Munzner, T. *Visualization Analysis and Design*; CRC Press: Boca Raton, FL, USA, 2014.
29. Khronos. GL Transmission Format. Available online: https://www.khronos.org/gltf/ (accessed on 5 April 2020).
30. Nguyen, V.T.; Jung, K.; Yoo, S.; Kim, S.; Park, S.; Currie, M. Civil War Battlefield Experience: Historical Event Simulation using Augmented Reality Technology. In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), San Diego, CA, USA, 9–11 December 2019. pp. 294–2943. [CrossRef]

31. Davis, F.D. A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory Additionally, Results. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.
32. Davis, F.D.; Bagozzi, R.P.; Warshaw, P.R. User acceptance of computer technology: A comparison of two theoretical models. *Manag. Sci.* **1989**, *35*, 982–1003. [CrossRef]
33. Dishaw, M.T.; Strong, D.M. Extending the technology acceptance model with task–technology fit constructs. *Inf. Manag.* **1999**, *36*, 9–21. [CrossRef]
34. Li, Y.M.; Yeh, Y.S. Increasing trust in mobile commerce through design aesthetics. *Comput. Hum. Behav.* **2010**, *26*, 673–684. [CrossRef]
35. Verhagen, T.; Feldberg, F.; van den Hooff, B.; Meents, S.; Merikivi, J. Understanding users' motivations to engage in virtual worlds: A multipurpose model and empirical testing. *Comput. Hum. Behav.* **2012**, *28*, 484–495. [CrossRef]
36. Anonymous. Battle of Palmito Ranch Augmented Reality Demo Version 3. 2019. Available online: https://youtu.be/PH9rLrZxQhk (accessed on 1 June 2020).
37. Becker, D. Acceptance of mobile mental health treatment applications. *Procedia Comput. Sci.* **2016**, *98*, 220–227. [CrossRef]
38. Davis, F.D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.* **1989**, *13*, 319–340. [CrossRef]
39. Hwang, H.; Takane, Y. *Generalized Structured Component Analysis: A Component-Based Approach to Structural Equation Modeling*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2014.
40. Hwang, H.; Takane, Y.; Jung, K. Generalized structured component analysis with uniqueness terms for accommodating measurement error. *Front. Psychol.* **2017**, *8*, 2137. [CrossRef] [PubMed]
41. Jung, K.; Panko, P.; Lee, J.; Hwang, H. A comparative study on the performance of GSCA and CSA in parameter recovery for structural equation models with ordinal observed variables. *Front. Psychol.* **2018**, *9*, 2461. [CrossRef]
42. Jung, K.; Lee, J.; Gupta, V.; Cho, G. Comparison of Bootstrap Confidence Interval Methods for GSCA Using a Monte Carlo Simulation. *Front. Psychol.* **2019**, *10*, 2215. [CrossRef] [PubMed]
43. Hwang, H.; Jung, K.; Kim, S. WEB GESCA, 2019. Available online: http://sem-gesca.com/webgesca (accessed on 24 June 2020).
44. Shelstad, W.J.; Chaparro, B.S.; Keebler, J.R. Assessing the User Experience of Video Games: Relationships Between Three Scales. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting, Washington, DC, USA, 28 October–2 November 2019; SAGE Publications Sage CA: Los Angeles, CA, 2019, Volume 63, pp. 1488–1492.
45. Beshai, S. Examining the Efficacy of an Online Program to Cultivate Mindfulness and Self-Compassion Skills (Mind-OP): Randomized Controlled Trial on Amazon's Mechanical Turk. *PsyArXiv* **2020**. [CrossRef]
46. Tsai, J.; Huang, M.; Wilkinson, S.T.; Edelen, C. Effects of video psychoeducation on perceptions and knowledge about electroconvulsive therapy. *Psychiatry Res.* **2020**, *286*, 112844. [CrossRef] [PubMed]
47. Paine, A.M.; Allen, L.A.; Thompson, J.S.; McIlvennan, C.K.; Jenkins, A.; Hammes, A.; Kroehl, M.; Matlock, D.D. Anchoring in destination-therapy left ventricular assist device decision making: A Mechanical Turk survey. *J. Card. Fail.* **2016**, *22*, 908–912. [CrossRef] [PubMed]
48. Rauschnabel, P.A.; He, J.; Ro, Y.K. Antecedents to the adoption of augmented reality smart glasses: A closer look at privacy risks. *J. Bus. Res.* **2018**, *92*, 374–384. [CrossRef]
49. Venkatesh, V.; Thong, J.Y.; Xu, X. Consumer acceptance and use of information technology: Extending the unified theory of acceptance and use of technology. *MIS Q.* **2012**, *36*, 157–178. [CrossRef]
50. Williams, M.D.; Rana, N.P.; Dwivedi, Y.K. The unified theory of acceptance and use of technology (UTAUT): a literature review. *J. Enterp. Inf. Manag.* 2015, *28*, 443–488. [CrossRef]
51. Nguyen, V.T.; Dang, T. Setting up Virtual Reality and Augmented Reality Learning Environment in Unity. In Proceedings of the 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct), Nantes, France, 9–13 October 2017; pp. 315–320. [CrossRef]