3-2018

Virtual Agent Interaction Framework (VAIF): A Tool for Rapid Development of Social Agents

Ivan Gris
The University of Texas at El Paso, ivangris4@gmail.com

David G. Novick
The The University of Texas at El Paso, novick@utep.edu

Comments:
Technical Report: UTEP-CS-18-19

Virtual Agent Interaction Framework (VAIF): A Tool for Rapid Development of Social Agents

Ivan Gris
The University of Texas at El Paso
El Paso, Texas
ivangris4@gmail.com

David Novick
The University of Texas at El Paso
El Paso, Texas
novick@utep.edu

ABSTRACT

Creating an embodied virtual agent is often a complex process. It involves 3D modeling and animation skills, advanced programming knowledge, and in some cases artificial intelligence or the integration of complex interaction models. Features like lip-syncing to an audio file, recognizing the users' speech, or having the character move at certain times in certain ways, are inaccessible to researchers that want to build and use these agents for education, research, or industrial uses. VAIF, the Virtual Agent Interaction Framework, is an extensively documented system that attempts to bridge that gap and provide inexperienced researchers the tools and means to develop their own agents in a centralized, lightweight platform that provides all these features through a simple interface within the Unity game engine. In this paper we present the platform, describe its features, and provide a case study where agents were developed and deployed in mobile-device, virtual-reality, and augmented-reality platforms by users with no coding experience.

KEYWORDS

Virtual Reality; Agent Interaction Manager; Social Agents; Tools and Prototyping

1 INTRODUCTION

The complexity of creating embodied conversational agents (ECAs) [**?**], and systems with ECAs, has long been recognized (e.g., [**?**]). To date, the most comprehensive support for developers of ECAs has been the Virtual Human Toolkit (VHT) [**?**], an extensive collection of modular tools. Among these tools, SmartBody [**?**] supports animating ECAs in real time, and NPCEditor [**?**] supports building question-answering agents. The entire collection of VHT tools is comprehensive, sophisticated, and evolving. Indeed, VHT is explicitly aimed at researchers—in contrast to developers or practitioners—who want to create embodied conversational agents. Our own experience with VHT was that setting up and using VHT can be difficult for non-researchers.

Some tools for creating ECAs have been aimed more squarely at developers. For example, the UTEP AGENT system [**?**] enabled creators of ECAs to write scenes, with dialog and gestures, in XML, which was then interpreted at run-time and executed via Unity. Our experience with the AGENT system was that (1) writing in XML led to run-time errors and was difficult for non-coders and (2) the system's jury-rigged connection to Unity was prone to annoying failures.

What is needed then, from the perspective of both developers and newer researchers, is an authoring tool that (a) enables developers to build human-ECA dialogs quickly and relatively error-free, and (b) that runs inside Unity, to promote efficiency and reliability. Toward this end, we developed the Virtual Agent Interaction Framework (VAIF), a platform for rapid development and prototyping of human-ECA interaction. In a nutshell, VAIF is both an authoring system for human-ECA interaction and a run-time system for executing these interactions, all built within Unity.

This paper describes VAIF and its user interface, presents some initial applications, and discusses VAIF's limitations and planned extensions.

2 VAIF OVERVIEW

VAIF and its applications were developed to enable research into rapport between ECA and human, with particular emphasis on multimodal, multiparty, virtual or augmented reality interaction. It is suitable for use by developers and researchers and has been tested with non-computer-science users. VAIF provides an extensive array of features that enable ECAs to listen, speak, gesture, move, wait, and remember previous interactions. The system provides a few sample characters with animations, facial expressions, phonemes for lip-syncing, and a configured test scene. In addition, characters can be assigned a personality and an emotional state to enable the creation of more realistic and responsive characters.

VAIF is open source (https://github.com/isguser/VAIF), with a collection of short video tutorials (available at http://bit.ly/2FaL4bW). It can potentially be integrated with external interaction models or automated systems. VAIF requires Unity 2017.1 or later (it is backwards compatible with other versions, but functionality is not guaranteed), and Windows 10 with Cortana enabled for speech recognition. It is also possible to create an interface to other speech-recognition platforms such as Google Speech.

3 VAIF ARCHITECTURE

Consider the following interaction: a character points toward the horizon while saying "Look, dawn is upon us," while the sun rises. To implement this sort of interaction, VAIF works by creating timelines of events. Each character can appear in one or multiple timelines, and multiple characters can appear on the same timeline. A timeline defines things happening in the virtual world, and agents' interactions. For the example above, the character will be in the timeline with a pointing animation, the correct location to point and face to, the dialog and its respective lip-sync, and the sun moving while the sky changes color. While the character performs a major role and

is the sole interactive element, changes in the virtual environment have to occur in sync with the dialog and the interaction.

Developing a fully working human-ECA interactive system in VAIF involves five steps:

(1) Choosing one or more agent models from VAIF's character gallery. If needed, as explained in Section 3.1, developers can use tools outside VAIF, such as Fuse and Mixamo from Adobe, to create characters that can be used in VAIF.
(2) Integrating and configuring the agents, using VAIF's configuration screens.
(3) Writing the interaction in VAIF's interaction manager.
(4) Creating lip-synch for the agents. This is done through an external tool such as Oculus OVRLP.
(5) Putting the agents in a virtual reality world. This can be done with an external tool such as VRTK, a Unity asset.

In the following sections, we describe how to accomplish each of these steps, and we briefly explain how VAIF can be used with other platforms.

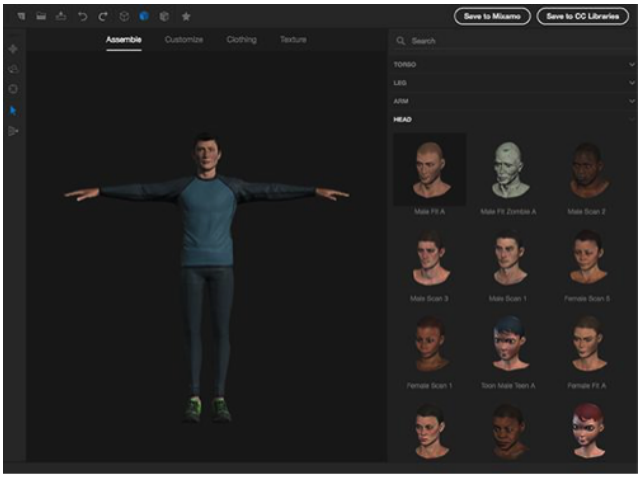

Figure 1: Fuse.

3.1 Model Setup

Agents, of course, are the main component of VAIF, which comes with a library of characters that can be used as-is with all of their features. Many developers will be able to build the systems they want with just these characters. Other developers may want to create new characters, and we encourage users of VAIF to contribute their characters to the library.

For developers who wish to create new characters, some setup is required to bring new models into Unity and VAIF. In the following paragraphs, multiple third-party software is provided as a reference; while we do not endorse or promote any of these applications, we have had a positive experience making use of them, and our sample scenes were developed with them.

Each agent is a 3D model, and while it can be anything, anthropomorphic characters are preferred. We recommend Adobe Fuse for character creation (see Figure 1). If a developers does not have the skills or the team to create 3D models for new characters, Fuse is a tool that will let them assemble the characters and clothe them via provided libraries, which can be extended. The tool also provides flexibility for facial features and body parameters, and it exports the character into an .fbx file, the 3D format required by Unity and VAIF.

Animations can be created using motion capture. Although systems like Opti-Track offer high quality captures, they can be prohibitively expensive or difficult to set up in the absence of a dedicated room. In those cases, applications like Brekel can be used to do Kinect-based motion capture.

Another option is to use Mixamo, also from Adobe, which lets developers rig and animate characters using their motion capture library. The downside of this approach is that some very specific animations will be hard to find.

Once the character is animated, the developer will need to add phoneme and emotion blend shapes. In this part of the process, the developer creates the vismes, which are the facial shapes that correspond to each phoneme, and all emotion faces. The process is different from animation, as it relies on manipulating the polygons directly rather than through the rig (bone) structure. A character without blend shapes will still work, but it will not perform the correct movements when speaking, and it will not be able to display emotions.

3.2 Agents

Once a character is properly set up, it can be integrated into VAIF.

The agent's emotional state and personality are among the first things to configure. While personality is a read-only variable, as personalities typically should not change during short periods of time (cf., emotions), it can be used to affect agent behavior in conjunction with the character's emotional state. For example, a happy extroverted character will move differently than a happy introverted character.

Emotions in VAIF are based on Plutchik's wheel of emotions [**?**] but can be modified to fit other emotional models. Likewise, personalities are based on the Myers-Briggs Type Indicator [**?**] but can be swapped for other personality models. Developers have access to the emotional state during runtime, so characters can be programmed to change their facial expression and dialogs after an emotion check (e.g., play a dialog with a happy or surprised tone while choosing a smiling face if the "joy" emotion is high) (see Figure 2). Emotions are not automatically mapped to facial expressions because VAIF does not display the exact blend of emotions in our face consistently and constantly. This can help create some interesting agents (e.g., an agent that smiles when it is angry).

Once the initial emotion and personality have been set, the character status is displayed (see Figure 3). Characters can be in any subset of the following states:

Speaking: An audio file is playing and the character is lip-syncing. Animations and movements can be played while on this state.

Listening: The character is waiting for a verbal response from the user. If this is enabled while the character is on the speaking state, the user can interrupt the character.

Waiting: Similar to the listening state, the character is waiting for an environment response or user action, such as a visual queue or a script execution (e.g. waiting for the user to grab an item with VR controllers).

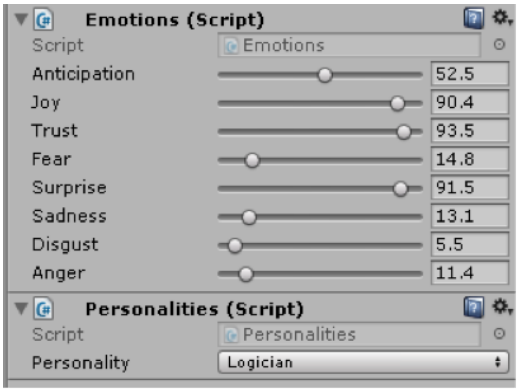

Figure 2: Agent emotional state and personality. Each emotion can be blended with others and vary in intensity. These parameters are used for conditional branching.

Figure 3: Character interaction states and managers. Any subset of states can be active at the same time with a few exceptions (i.e. cannot be waiting and speaking simultaneously).

LookAt: The agent is being looked at by the user.

In-Range: The character is close enough to the user to initiate interaction.

Moving: The agent is performing a translation in virtual space, such as walking or following another character.

The states are used internally by the system to detect different situations. For example, if the character is in the speaking state and the user talks, the character knows it is being interrupted and can react to it. If the character is waiting, nothing happens; but when the flags for being in-range and being looked-at turn true, the character will switch from waiting to speaking because it found an interlocutor. These states are calculated automatically depending on the interaction state.

3.3 Interaction Manager and Events

The interaction manager is the central piece of the VAIF architecture. A manager is a script that requires no parameters but enables the characters to speak, move, gesture, remember, and look-at. It

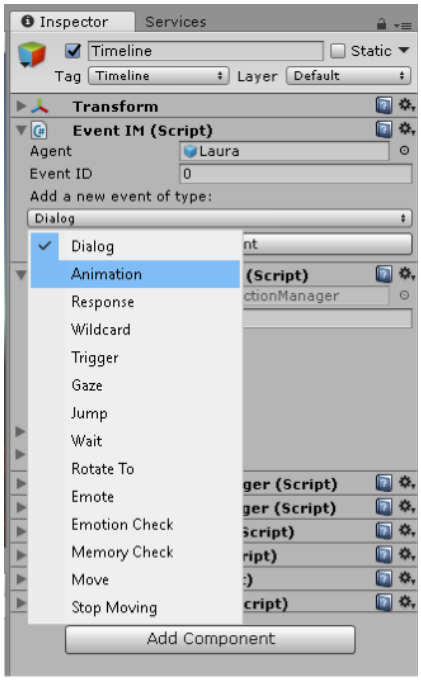

Figure 4: All the events developers can choose from through the interaction manager. Events can be assigned to one or multiple characters.

controls the flow of events across the interaction by creating a sequential timeline of events. VAIF's timelines contain all the events and control all characters' actions. In other words, the interaction manager enables developers to create events that are easy to configure and are applied automatically to the characters (see Figure 4).

The system can create the following events:

Dialog: Enables the user to specify an audio file or the name of the audio file, the character that will say it, a transcript of what is said in the audio file, and what to do (go to a specified event, potentially another dialog) if the character is interrupted (see Figure 5). Dialog automatically connects the audio file or audio file name to its respective audio and lip syncs while playing the file.

Animation: Enables developers to specify by name which animation to play and which character should act. It includes options to delay the animation or to stop the animation from playing after a certain amount of seconds. It also allows developers to loop the animation (useful for walking, breathing or idle animations).

Response: Puts the specified agent in a listening state and enables the user to speak. It also enables developers to specify through text (a) the words to be recognized and (b) the event index in the timeline to which each recognized keyword or phrase should lead. It also provides options to timeout the agent's listening state, where if no word is spoken in a certain number of seconds, the timeline leads goes to another specified event. Finally, it provides a feature where, after the specified number of misrecognitions occur, the timeline moves to another developer specified event, which could

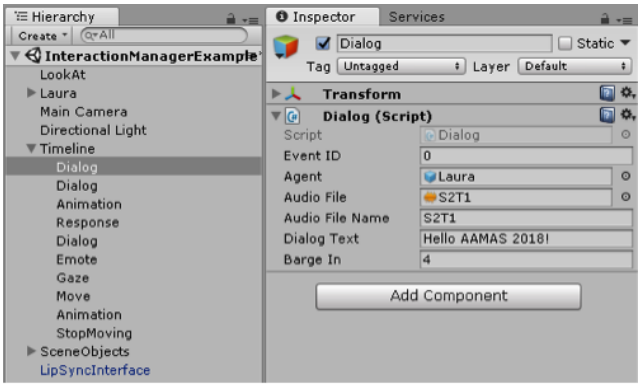

Figure 5: The first event, a dialog, from a timeline. By dragging and dropping or typing in the different fields, the character Laura will say and lip-sync to a dialog, in this case, "Hello AAMAS 2018!" If interrupted, the timeline jumps to event 4, in this case, another dialog.

be a dialog explain again what is expected from the user—or a very frustrated animation from the character.

Wildcard: In certain situations the developer may want user feedback in the form of speech, either to improve engagement or to gather information to which the agent need not react. In these cases, a wildcard is used to have a character pause the interaction while it listens to the user. This is useful when the agent is asking for things like names, places, dates, or other things that would require extremely long lists and events for recognition and response. In this case, the agent listens to the response, annotates it as dictation, but does not react. Imagine the following interaction:

```
Agent: What is your name?
User: My name is Mike.
Agent: Nice to meet you. I'm Laura.
```

In this example, the agent used a wildcard to record the user's response, but the agent's reaction is the same for everyone regardless of their name.

Trigger: This is a special event that does not affect a character in particular but rather the environment or entire scene. Triggers call other functions within Unity at runtime. Consider the following example: if I have a function that creates rain, I may want to call it after an agent dialog stating what a nice weather they have today. In this case, the developer does not know the exact time in which this dialog will be played. By using a trigger, developers can specify the game object, the component, the method, and the parameters that they want to execute, in this case, when to activate the rain and at what intensity. Because this requires C# coding experience and Unity experience, it is considered an advanced feature.

Gaze: This event enables developers to specify an interest point for the agent's gaze. Gaze can be specified by a general direction (i.e., up, down, left up, down right, etc.) or towards an object or agent. Object gaze is unconstrained, so if the object of interest is located behind the character it will roll its eyes backward and stare into its brain. Future updates will address this by making the character rotate its head and body, if appropriate, to direct its attention to the point of interest using the model developed in [**?**].

Jump: Jump events do not refer to agents performing a physical jump. Instead, jump is used as a branching condition in the timeline where, upon reaching this event, the timeline will go to a specified event. This is useful in cases where developers want a looping interaction, where once the timeline reaches the end it jumps back to the first event.

Wait: If an agent is specified, then the agent goes to the wait state for the specified number of seconds. If no agent is specified, all events in timeline and all characters wait for the specified amount of time. It is also possible to wait for an indeterminate amount of time and exit that state by using a trigger event.

RotateTo: This event reorients the agent toward an object or another agent. An object from the scene can be dragged and dropped into this field, and the character will know in the direction toward which it should reorient.

Emote: Emote has two functions. First, it enables developers to change the emotional state, that is, to alter the values of any of the emotions shown in Figure 2. This can be done by adding or subtracting to the current value or by completely overriding them. In addition, this tag can be used to make the character display a different facial expression. It is important to note that the emotional value does not need to correspond to the facial expression. Likewise, developers can opt not to change the emotional values or the expression at the same event. All emotional values range between 0 and 100. Trying to increase or decrease a value below or over this range has no effect after reaching the minimum or maximum limits.

EmotionCheck: This event enables developers to specify an emotion and check the value. Developers can then choose to jump to a different event depending on their condition, which can compare the emotional value to a predetermined one (e.g., if joy is greater than 75 and trust is greater than 80, go to event ID 3, otherwise go to event ID 7).

MemoryCheck: Memory is a list of event IDs. Every time an event takes place it gets added to the list. Within this list, developers can check if particular events have been triggered. Saving memories is an automated operation that enables developers to include events that can check if an event, or a set of events, was visited and then jump. Because all events have an ID, memory can check if a character looked at something, said something, moved in a certain way, had an emotional change, or even what the user responded throughout the interaction. Memories can also be saved and carried across Unity scenes or loaded at a later time should the application be suspended. Because these are saved to a file, memories are permanent, and developers or experimenters can load them selectively, with a memory file per participant for example.

Move: Moves the character to the specified position. The positions can be invisible game objects that can be dragged into the position field by the developers. It also has a *follow* option, enabling characters to track and move towards an object continuously. An interesting use of this feature is to have an agent follow a walking user in room-scale VR.

StopMoving: If the character is moving towards a static object, it will stop automatically once it reaches its destination. However, this tag enables developers to stop moving while still approaching the target or to stop following an object or user if the option was

enabled in the move event. This event has no effect if a character is not moving.

Looking at the overall function and use of the interaction manager, the simplest use case is to create a scripted sequence of events. Even when all events have been predefined, the path is not known until runtime. This means that all character's dialogs are created as events before running the application, but there is branching depending on user responses, emotional states, memory checks, and triggers. Because agents keep track of when users are within range and being looked at, the interaction is not necessarily linear, as different users may approach characters in a different order.

A more complex approach is to have multiple timelines, each with its own events, so that some timelines are shared between multiple characters, while some are individual and can be approached in a specific order or left up to the user.

Furthermore, because all events are meant to connect internal agent functionality and Unity features for agent developers through a simplified interface, advanced developers can skip the interface altogether and connect other dialog systems, gaze models, or other resources by interfacing with the timelines, creating and adding events during runtime produced by their systems. We note, though, testing and integration of these features has been limited.

Consider the following interaction alongside the events required to perform it: the user approaches a character and looks in its direction.

```
[RotateTo] User
[Animation] Talking
[Gaze] Down
[Gaze] Up
[Gaze] User
[Dialog]: Excuse me, who are you?
[Wildcard]
[Animation] Excited
[Dialog]: Wonderful, nice to meet you.
Can you help me carry these boxes?
[Response]Yes/No
[Dialog] (If yes): Great, I'll grab one.
You can take the other.
[Wait] Until box is grabbed.
[Animation] Grab box.
[Move] Box pile
[Emote] Joy = 100.
[Dialog] (If no): I'll find help elsewhere.
Thanks for nothing.
[Move] Inside building
[Emote] Anger = 100.
```

In this example, developers can use just a few events to specify how the character will move, what will it say, and how will it react to the user. At a later time, the emotions can be used to respond to the user if he or she approach the character subsequent times.

3.4 Lip-Sync

Lip sync is performed through the Oculus OVRLP module (see Figure 6). This system runs independently of the target platform, meaning it works for any device.

OVRLP uses 15 different facial shapes that are mapped to different phonemes. The audio file gets parsed in search for phonemes, and the shapes blend, thus creating the lip sync effect. A sample

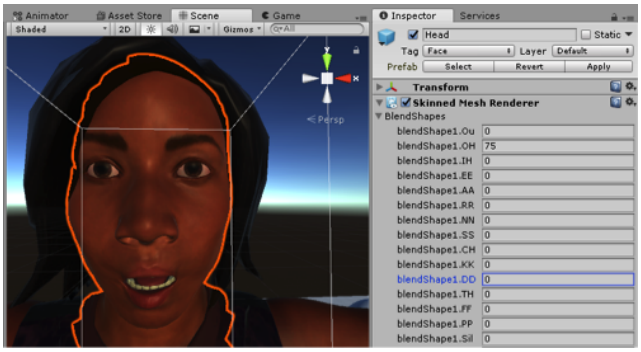

Figure 6: Character displaying the "OH" phoneme blend shape at 75%.

configuration is provided in the VAIF package and it is ready to be included on new characters.

3.5 Room-Scale Virtual Reality

To port the agents into VR systems, we recommend VRTK, a Unity asset that enables developers to build scenes for multiple platforms, including Oculus and HTC VIVE.

There are two elements that are particularly important with these setups: (1) LookAt, and (2) InRange. The first component is an invisible long box that acts as a ray cast, meaning it can detect where the user is looking if it is attached to a camera. This module works with any type of Unity camera, including VR and AR cameras. The latter serves as an invisible range that can be configured to represent how close the user is to the characters. This module detects movement in room-scale VR, so the user can approach a character by walking towards it, and the system will know when the user is close enough. Distances can be changed to fit the appropriate scale. Developers can use this component to have the user interact with characters while approaching and looking at them.

3.6 Augmented Reality and Other Platforms

Projects can be compiled through Unity for a variety of platforms, including iOS or Android devices. AR headsets usually provide a Unity developers kit as well that can be installed as a Unity package. Although the development process remains virtually the same up until compilation, the speech recognizer provided with VAIF works only in Windows 10.

4 APPLICATIONS

VAIF has been used in commercial and research applications. In this section we present three cases: a commercial game currently in development, a room-scale VR application with multiparty interactions, and a minimalistic agent for AR and mobile devices.

4.1 Room-Scale VR and Multiparty Interaction

Two VR applications have used VAIF to create embodied conversational agents with social capabilities.

The first application is a commercial fighting game. While for the most part players go through endless hordes of enemies using

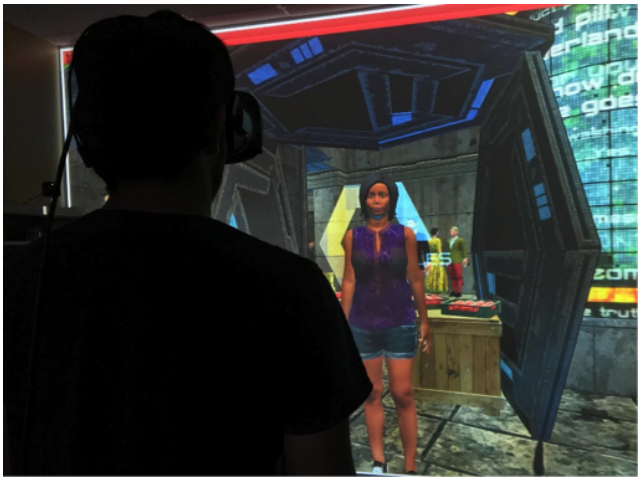

Figure 7: Room-scale VR project using VAIF, where the user walks up to the character to initiate the interaction.

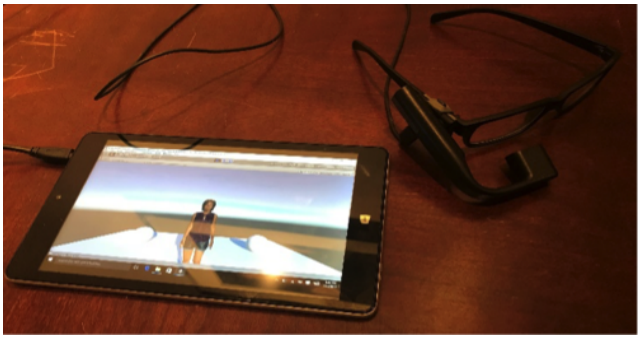

Figure 8: A VAIF agent on a Windows tablet connected to another AR device.

their fists, swords, and ninja stars in a futuristic dark city, when the players face the main characters VAIF handles their conversations. This enables the game to use speech recognition to upset the enemies, to ally with them, to convince them to join the player's cause, or to stall the fight for a few moments, effectively changing the game's outcome. This application uses VAIF's memory modules as well as response, dialog, animation, and emotion events.

The second application involves a research project located in a futuristic marketplace. The goal of this project is to examine and experiment with user initiative in multiparty non-linear interactions (see Figure 7).

4.2 AR and Education

Most agents deployed for educational purposes are not meant to replace the instructors but rather to enhance the students' learning experience. VR and AR educational experiences typically include detailed visuals of the environment (e.g., the solar system, a cell structure, a 360 documentary video, etc.) but provide limited interaction. With VAIF, agents can explain the information within their

Figure 9: A character on an iPhone, using Mira, an AR headset that mirrors the phone display.

context, request feedback, and evaluate learning outcomes (e.g., [?]).

Another approach is to help instructors create their own virtual teaching assistants, who reduce the teacher's workload and help him or her focus on the class while reaching out to larger student groups in an engaging way. Agents have also been developed using a more minimalistic approach for augmented-reality lenses, such as Mira, and mobile devices. A professor of education has, for example, has used VAIF to implement a question-and-answer agent-based applications on a Windows 10 8-inch tablet (see Figures 8 and 9).

5 CONCLUSION

In this paper, we presented a lightweight platform that enables developers and researchers to create their own ECA applications with speech recognition, (virtual) spatial awareness, and interaction control. We described all of the system's capabilities and explained how VAIF is being used to develop characters for multidisciplinary research and games.

5.1 Limitations

One of the major limitations of VAIF involves the difficulty of integrating external technologies into the characters. Integration of independent stand-alone models coded in different languages (e.g., back-channels, turn-taking, prosody, discourse managers) are particularly difficult, as they must either be compiled within Unity and C#, or connected through a network and message the input and output across applications. Although it is possible for us to integrate some of these systems, the complexity and system knowledge required to do so makes one of the most-requested features unusable for average users.

Another limitation is the speech recognizer, which is restricted to Windows 10 devices. Although this is usually enough for self-contained applications, additional options are needed for cross-platform compatibility. The research team has connected Google Speech and IBM Watson to precursors to VAIF, but this requires additional external programs and network connections that place similar or worse restrictions on cross-platform use.

Finally, even when extensive written and video documentation is provided, to fully exploit VAIF's capabilities users need to create several timelines and moderate the flow of the interaction through

events (i.e., prevent events or conversations from happening if other events or conversations with different characters have not occurred yet), which requires familiarity with the system and knowledge about social agents.

A tool, regardless of how good it is, does not necessarily create great interactions and ECAs. Just as a great game engine does not guarantee great games, our research efforts focus not only on the tools, but also character, dialog, and interaction design. This involves gameplay development, usability testing, and trial-and-error. Virtual agents are a multidisciplinary collaboration among behavioral scientists, writers, artists, voice actors, and programmers. The tool is meant to provide a shortcut for most of the technical implementation details, but socially aware agents go well beyond that.

5.2 Future Work

VAIF is the result of five years of research and over a dozen virtual-agent applications. It is an important step towards an easy-to-use social-agent development tool. We are currently using VAIF to create multi-agent applications as test-beds for studying human-ECA interaction.

At this point, most of our efforts involve usability testing and enhancing existing features to have a stable core system before more features are integrated. But VAIF could be enhanced and expanded in several key respects. We plan:

- To provide additional speech recognizers and integrate them to the extent possible to improve cross-platform compatibility.
- To simplify the visualization of the events through tree representations of timelines to make branching easier to observe and design from the developers? perspective.
- To develop a co-op mode, where multiple people can experience and interact within the same virtual environment and intelligent virtual agents.
- To provide additional speech recognizers and integrate them to the extent possible.
- To simplify the visualization of the events through tree representations of timelines to make branching easier to observe and design.

We look forward to sharing these enhancements as they are developed with the VAIF user community.

ACKNOWLEDGMENTS

We thank Adriana Camacho, Diego Rivera, Alex Rayon, and Michelle Afravi, Julie Hinojos, and Aaron Rodriguez for their contributions to the development of VAIF.