Editor: Mike Potel
www.wildcrest.com

A Virtual Environment for Teaching Social Skills: AViSSS

Justin A. Ehrlich and James R. Miller, ***University of Kansas***

Being a teenager in today's world is tough. Physical changes, academic demands, and peer pressure impact teenagers' development, shaping who they are and who they become. Even though middle school and high school curricula don't focus on social-skill development, much of what our children learn regarding social behavior is in the context of school or its extracurricular activities. For many typical adolescents, the development of appropriate social skills can be demanding. For those with Asperger's syndrome (AS), an Autism Spectrum Disorder (ASD), mastering social skills is much harder and can be a determining factor in the individual's school success.

The challenge confronting individuals with AS, their families, and their educators is how to teach appropriate social skills in an environment where these skills can be generalized. Equally important is providing an educational experience in which individuals with AS aren't ostracized but are made to feel safe, even when they fail to make appropriate choices in a given social situation.

Students with AS are often visual learners, and research has shown they can learn social skills through computer-based exercises. Researchers have suggested virtual environments as an effective way to teach social skills to individuals with ASDs.[1] Using a virtual environment as a safe zone to teach social skills can provide opportunities to practice various situations without much tension or anxiety. Toward that end, we've developed AViSSS (Animated Visual Supports for Social Skills), a 3D virtual environment.

The Challenges of AS

AS is a developmental disability defined by impairments in social interactions and restrictive, repetitive patterns of behavior, interests, and activities. Individuals with AS lack appropriate social skills, have a limited ability to take part in reciprocal conversation, and don't seem to understand many of the unwritten rules of communication and social conduct that their peers seem to naturally learn through observation. These characteristics significantly impact their ability to demonstrate social and emotional competence, including self-awareness, controlling impulsiveness, working cooperatively, and caring about others.[2]

A primary social-skill concern for adolescents with AS is their inability to select appropriate problem-solving skills to handle a situation. Defining the problem accurately can be the first step in solving problematic situations. Unfortunately, the lack of understanding about cause and effect makes problem-solving difficult. In addition, problem-solving skills must be generalized in various ways across settings and people. For an individual with AS who has restrictive patterns of interest and behaviors, applying rules across settings, people, and situations is quite problematic.

For example, students with AS might memorize sequences or information but might not be able to use that knowledge when needed. They might also be able to apply their skills in one situation but not in a different setting or with different people.

Choosing a Rendering Engine

Before we implemented the project, we reviewed a variety of existing environments to select a starting point that wouldn't force us to start from scratch.

Second Life

One of the first environments we considered was Second Life (https://join.secondlife.com), a virtual social world anyone can join and in which any organization can "set up shop." We quickly discovered two major unsolvable issues: the inabilities to protect students in an uncontrolled environment and to remove any social consequences in an inherently social environment.

Game-Based Alternatives

Another alternative was a traditional game engine that handles 3D rendering, animations, and operating logic. Most engines follow the same model, which is to allow the developers to create

 Published by the IEEE Computer Society

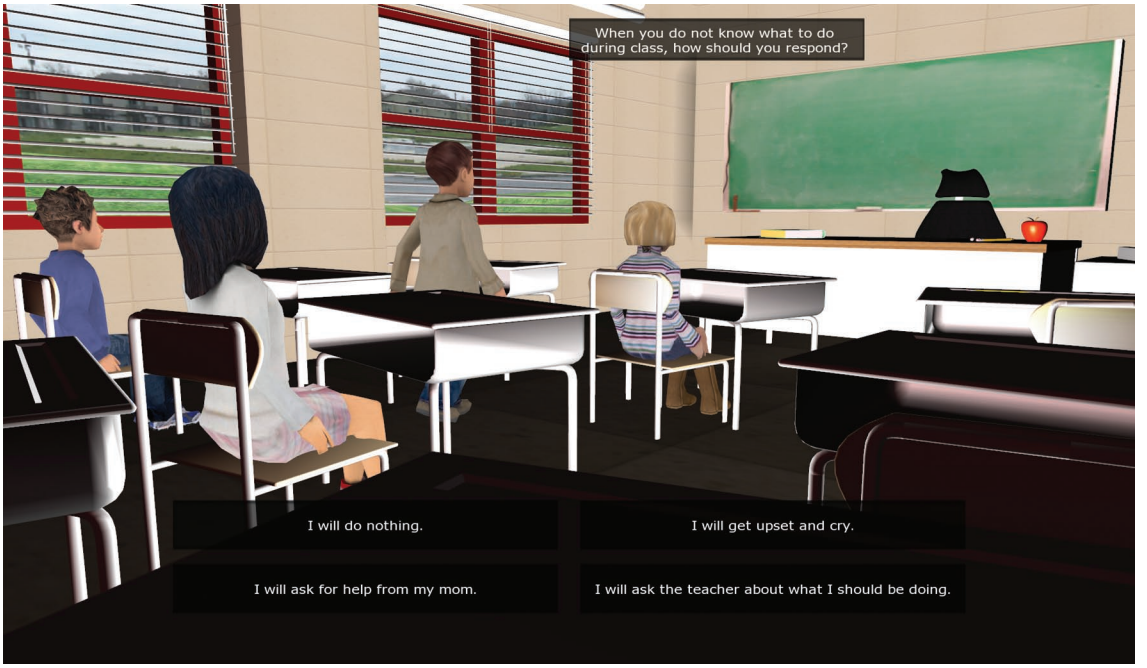


Figure 1. A situation from the classroom environment in AViSSS (Animated Visual Supports for Social Skills). Here, the user is sitting at his or her desk, presented with a situation and a choice to make. For the scene to continue, the user must select one of the presented possibilities.

a scripted environment that can be passed to the game engine logic. Because games incur high development costs and produce significant revenue, modern game engines are expensive.

Often, older-generation engines are open source or free for academic use, but full functionality often comes at the expense of inferior graphics. We considered using an open source solution such as id Tech 3, which powered Quake 3 Arena (www.idsoftware.com/business/techdownloads). However, we were hesitant because it's older and doesn't take advantage of the latest hardware or support high-resolution textures on high-resolution models. Likewise, missing code from release's GNU General Public License (GPL) version, particularly the required skeletal animation system, limits development options.

We also considered free-for-academic-use engines such as Unreal Engine 2 (http://udn.epicgames.com/Two/UnrealEngine2Runtime.html), which powered Unreal Tournament 2003. However, only part of that code is open. Although the engine allows for extensions and some modifications, the full source is expensive. We need full control of the application, including the logic of the menu and the game itself. Another concern was licensing. We wanted as few restrictions as possible, but using the Unreal Engine 2 would bind us to the noncommercial clause or force us to purchase a costly license.

Finally, much of the sophistication of game engines deals with collision detection and other simulated physics. AViSSS needed none of that.

OGRE

We finally chose OGRE (Object-Oriented Graphics Rendering Engine; www.ogre3d.org), a 3D rendering engine that abstracts the complexities of rendering 3D meshes and animations from low-level 3D APIs. There's absolutely no application-specific logic tied to OGRE; instead, its design lets you build an application on top of its rendering engine. This gives us the flexibility to write AViSSS as we see fit, while delegating the graphics to OGRE.[3]

Another attractive feature is the licensing; OGRE uses the GNU Lesser General Public License (LGPL). As long as we keep AViSSS separate and dynamically linked to the OGRE libraries, we can distribute it however we see fit. We want the options of releasing our software as open source, keeping it closed and proprietary so that we have exclusive distribution rights, or a mixture of both. Another advantage is that OGRE works on all three of our target platforms: Mac OS, Windows, and Linux.

AViSSS

We designed AViSSS to simulate everyday real-world situations. AViSSS has a collection of virtual environments such as hallways, restrooms, and cafeterias. Environments have multiple scenarios; a scenario can have several situations, each involving a problem the student must address. For example, in the gymnasium scenario, a user who dislikes exercise must first choose whether to participate in a physical activity. Once the user chooses the correct response (that is, to participate), he or she must deal with the overwhelming noise and commotion.

Scenarios are basically decision trees encoding social narratives. Each nonterminal node in a decision tree represents a choice to make. In some situations, such as in Figure 1, the student must choose a behavior; in others, such in Figure 2, the student must select an object. Typically, only one such

(a)

(b)

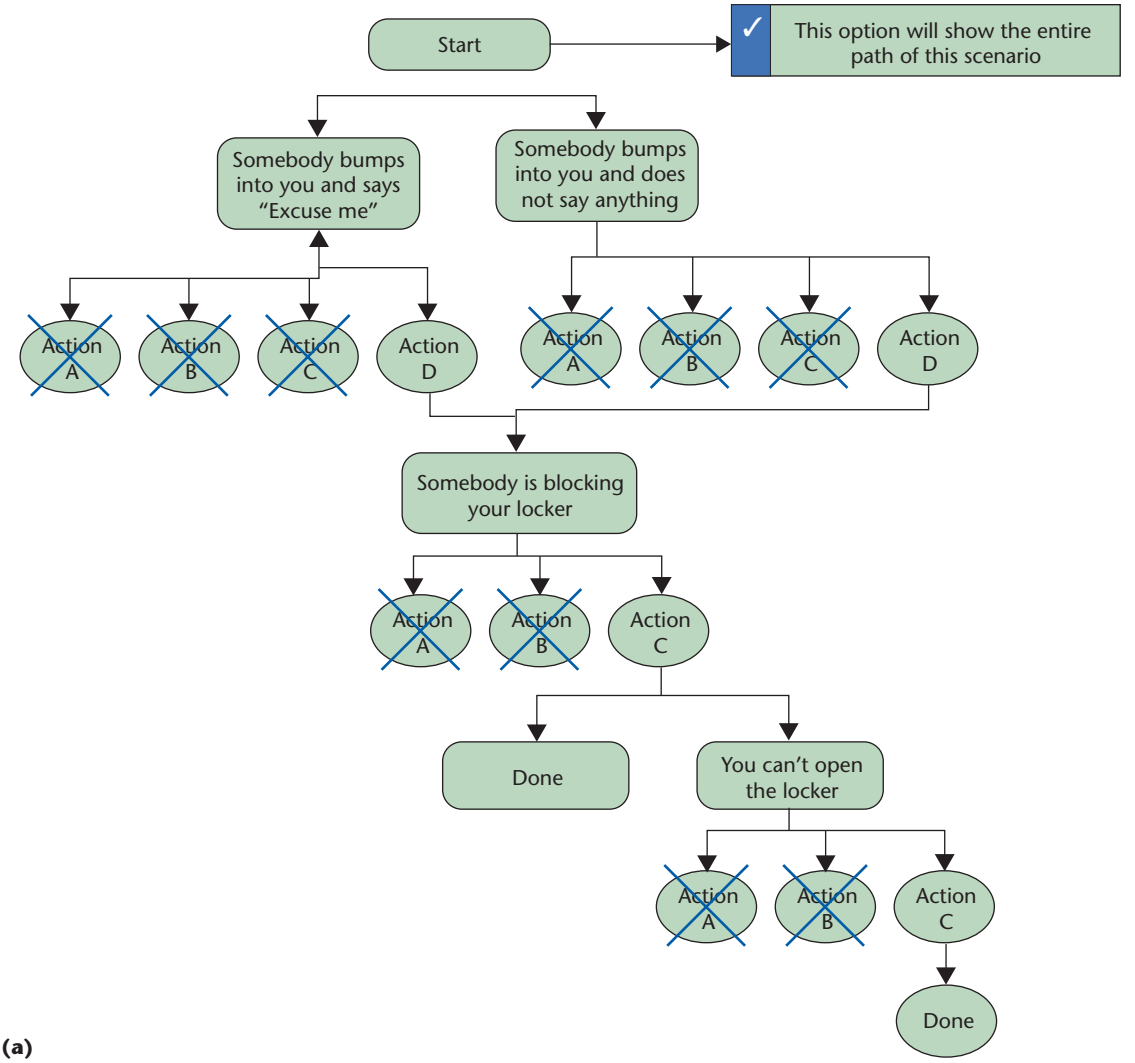
Figure 2. Two situations in which the user selects objects with the mouse. (a) In the bathroom situation, the user must select the empty stall that's clean and not out of order. (b) On the school bus, the user must pick an appropriate seat after learning that the user's usual seat has been taken.

choice is appropriate. Making choices advances the student through the tree. Leaf nodes represent the final outcome of a given set of decisions.

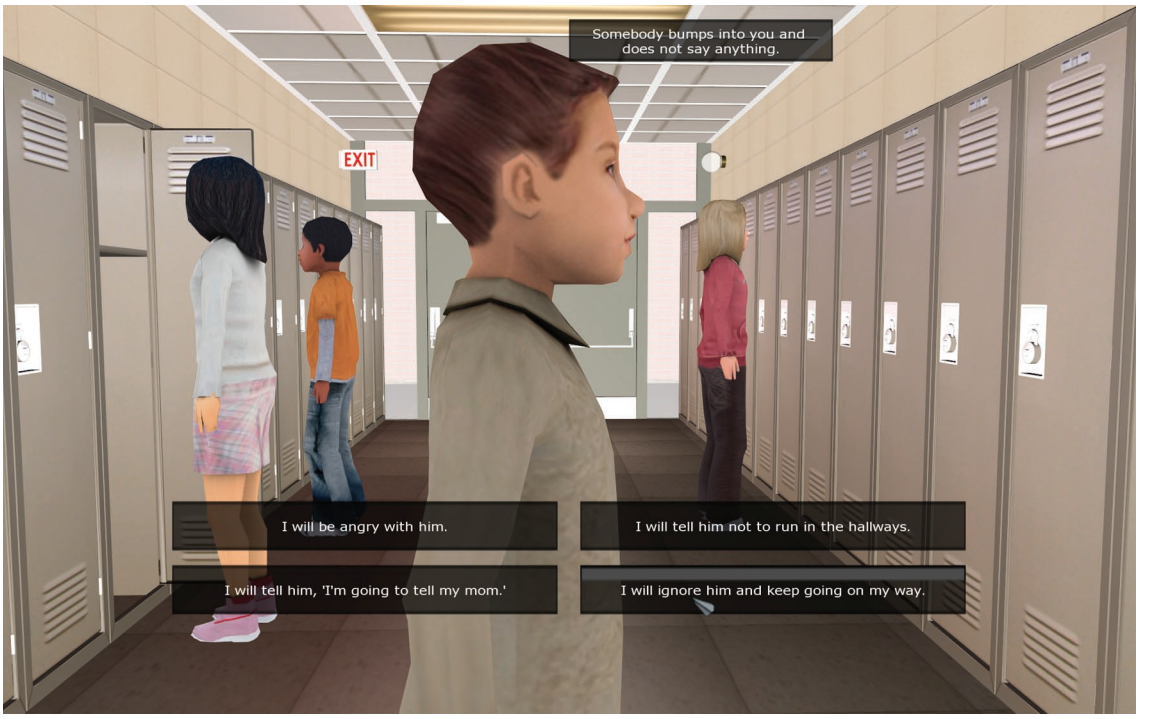
Figure 3a shows a decision tree for a hallway scenario; Figure 3b is a screenshot of this environment in AViSSS. The program presents four possible responses, one of which is clearly the best.

If the user makes a wrong decision, the application explains why that decision was poor. The student sees textual dialogue and hears a prerecorded message. Then, the last scene replays, and AViSSS asks the user to make a better decision, this time with the previous decision grayed out.

When the user makes the correct decision, the application selects the next node, and the environment changes to start the next situation. The application then displays a new list of alternatives. This continues until the application reaches the path's end. The application then displays the student's score, accompanied by verbal feedback, and lets the student



(a)



(b)

Figure 3. The hallway locker scenario: (a) decision tree and (b) screenshot. The screenshot shows AViSSS at the left child of the decision tree's root. The decision tree is traversed depending on the user's decisions.

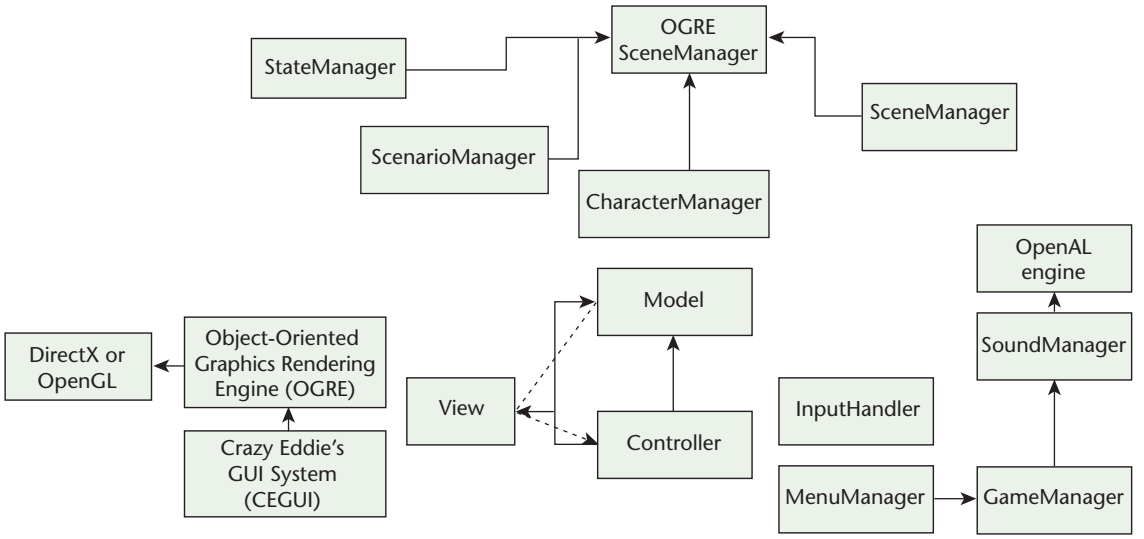


Figure 4. A Model-View-Controller (MVC) diagram of AViSSS. The MVC at the bottom center represents the corresponding objects and their support objects.

continue to the next scenario, which might be in another environment. For instance, after the hallway, the student enters the restroom scenario.

Our experts precisely picked each problem in each scenario to deal with a specific problem area for adolescents with AS. We've categorized each problem for administrative purposes. AViSSS lets the administrator (for example, a teacher) running the application choose the problem categories. The system records every response to give feedback to the administrator on the student's progress. This gives the administrator as much control and information as needed to successfully help the student.

Technical Design

Figure 4 illustrates our basic architecture, derived from the standard Model-View-Controller pattern.[4] OGRE is a view component, whereas the OGRE SceneManager is the model. The main controller runs a game loop, each pass of which gathers and processes user input, updates the corresponding model, and finally asks the view to render the models. At each time step, the models move and synchronize the camera, animations, objects, and the overall environment. Crazy Eddie's GUI System (CEGUI; www.cegui.org.uk) is an OGRE plug-in that uses OGRE to render the user interface. The OGRE view, when called by the controller, renders everything in the SceneManager and CEGUI.

The models keep track of the data constituting the scene. They store the scenarios, the current location in the current scenario, camera positions and angles, characters, 3D mesh object pointers, and animations. The models all sit on top of the SceneManager, which consists of the current 3D meshes and their positions. Each model that controls a 3D mesh object contains a corresponding managerial object with a pointer to that 3D mesh object in the SceneManager. The SceneManager only manages static meshes that make up the scene at a given frame; it doesn't contain any logic for that scene. The OGRE view can only render the scene as stored by the SceneManager. For a mesh to have dynamic functionality, such as animation or movements, its model's managerial object must calculate the changes and modify the mesh's positions and frames in the SceneManager.

For instance, if a character is walking in a certain direction, the CharacterManager model keeps track of the corresponding character object's angle, speed, current position, animation, and pointer to the 3D mesh, whereas the SceneManager stores the actual 3D mesh and position. The GameManager will call the CharacterManager to increment all the characters at a certain time. Then, the CharacterManager will use each character object's current position, angle, velocity, and animation frame to update its next position, angle, velocity, and animation frame. Next, each character object modifies the corresponding 3D mesh in the SceneManager using the new animation frame, angle, and positions.

Our controller consists of two basic input controllers and three support controllers. The first basic controller is the InputHandler, which processes keyboard events from the OGRE view. It directs the SceneManager to change the user's direction or position accordingly.

The second is the MenuManager, which controls the menu model. It handles menu choices and user decisions, both coming from the CEGUI view. When it receives menu choices, it updates its menu accordingly from the various other models.

When the MenuManager receives a decision, it first calls the GameManager to change the environment. The GameManager takes the decisions and queries the StateManager to determine which actions to perform on the basis of the script. The StateManager returns the actions to the Game-

```
<state index="2" expression="neutral" speak="Somebody bumps into you and
   does not say anything." action="boy01 delay 1 animateEnd walk animateOnce Idle1
   showButtons" decisions="4" skill="1" situation="0">
     <desc index = "0" input="1" description="I will be mad at him." output="5"
        value="3" animationsOnSelection="Explain1;Talk"
        soundOnSelection="Teacher_Talk1.ogg" descriptionOnSelection="He did not
        mean to bump into you, so you shouldn't get mad."/>
     <desc index = "1" input="2" description="I will tell him not to run in the
        hallways." output="5" value="-2" soundOnSelection="Teacher_Talk2.ogg"
        animationsOnSelection="Explain1;Talk" descriptionOnSelection="It is not
        your place to give orders."/>
     <desc index = "2" input="3" description="I will tell him, 'I'm going to tell
        my mom.'" output="5" value="-1" soundOnSelection="Teacher_Talk4.ogg"
        animationsOnSelection="Explain1;Talk" descriptionOnSelection="You must
        work out your own solution, besides your mom is not here."/>
     <desc index = "3" input="4" description="I will ignore him and keep going on
        my way." output="6" value="0"/>
</state>
```

Figure 5. The script portion (in XML) implementing the root's right child in Figure 2a. We developed our scripting language specifically for a decision tree model.

Manager, which parses and interprets them to determine the next necessary steps. Each action is a command to change something in the model. A correct decision might change the user's orientation (SceneManager), switch to the next situation (StateManager), or move and animate a character to a different location (CharacterManager). The GameManager uses these commands to call the various models to update their data. It sends any new menu changes back to the MenuManager, which updates its decision model accordingly.

The MenuManager then tells the CEGUI view to update its display. The OGRE and CEGUI views render themselves after each game loop increment if the SceneManager undergoes changes. The OGRE view uses the SceneManager's mesh data structures to gather the latest scene created by the models to render the world appropriately using OpenGL or DirectX. To render the menu on top of the OGRE view, the MenuManager uses its own SceneManager instance, which contains a user interface object.

Additional Libraries

We tried to use existing libraries as much as possible so that we could focus on our application's functionality. Because OGRE is only a graphics renderer, we had to include a few more libraries for needed functionality. For example, we needed an audio engine for sound effects and dialogue. We chose OpenAL (http://connect.creativelabs.com/openal/default.aspx), a cross-platform API for sound that conveniently uses the LGPL. OpenAL can play Ogg files natively. Ogg files are like MP3s, except they use open source technology.

We also needed an XML reader. We needed something simple to let us read the pathway scripts, and we didn't want to write our own XML parser. For this, we used TinyXML (www.grinninglizard.com/tinyxml), a basic XML parser written by Lee Thomason licensed under the zlib license.

Scripts

For each scenario, AViSSS follows a script. The scripts, created by AS specialists Hyo Jung Lee, Sheila Smith, Sean J. Smith, and Brenda Myles, are based on prior research on pathways. Each script services a number of targeted skills within an encompassing environment.

The scripts use a custom-built language describing the path to follow and telling AViSSS what to render, which models to load, which sounds to play, and which animations to use. This allows AViSSS to be completely customizable for any situation or environment. The scripts initialize the environment by creating lights, loading models, and positioning the camera. Our scripting language's logic lets users follow different paths (or reverse the current path) on the basis of their choices.

Each script consists of an XML file. Each situation is represented by an XML element and a named state, and is numbered with an ID. Figure 5 shows the script portion implementing the root's right child in Figure 2a.

Each state contains the decision buttons' text along with the corresponding 3D scene. Each decision is a branch telling the scripting engine which state to select next. If the user chooses the correct decision, the application proceeds to the next state.

Although our implementation of the script is deterministic, the script doesn't have to be. We've designed the scripting engine to allow for more-complex, nondeterministic paths. For example, if a child made a wrong decision, AViSSS would present one of several paths, each with a different consequence and state.

The Advice Center

Children with AS are in great need of an *advice center*—an inner voice intentionally formed to make daily decisions. However, they have difficulty forming one without external assistance. Once

they do form one, they consult it for correct decisions for daily situations. This querying requires extra mental processing, compared to a typical adolescent using innate problem-solving skills.

To help students form advice centers, AViSSS incorporates a simulated advice center in the form of an iconic image and prerecorded narration talking through every decision. Once the user makes a decision, the advice center explains why or why not the choice was correct. At the end of every scenario, the advice center provides a brief summary connecting the school situation with additional examples outside school to generalize the learned skills.

Evaluation

We completed a formative evaluation of AViSSS involving a team of AS researchers, visualization experts, adolescents with AS, and their parents. The tests were all qualitative in dealing with the functionality and the content of the materials. We distributed the AViSSS application to the team to review and interact with. We asked them questions regarding their initial response and the application's perceived effectiveness. A meeting was held for the team to voice any concerns, insights, questions, or feedback. This not only let us gain insight from AViSSS's intended target audience but also helped the users better understand our direction and goals. This evaluation has been very positive in that we're hitting on the most problematic areas for adolescents with AS. Feedback regarding the interface has led to changes that better accommodate our subjects.

One notable change has been the addition of the advice center. Originally, a virtual teacher interacted with students when they made an inappropriate choice. We soon learned that students with AS generally didn't respond well to that teacher. They appeared to perceive teachers as being uninterested, impatient, and ill equipped to deal with them. As a result of the formative evaluation, we decided to design decision feedback to foster advice centers.

Furthermore, project staff are about to implement a multiple-baseline research study to examine this tool's effectiveness. In particular, this study will try to answer three questions: Does AViSSS effectively teach social skills to individuals with AS? Do skills learned via AViSSS generalize to the school and community setting? Do participants view the tool as an effective, efficient tool for social-skills training?

Depending on our initial offerings' success, we might add nonschool environments—for example, a mall, a movie theater, home life, or a public pool. We designed AViSSS and its scripting language to enable plug-ins allowing new environments, content, and functionality. We'll develop and release a scenario-and-level editor, with which users can create their own environments, such as the student's own school or home.

We want to eventually add a module allowing a more user-controlled experience. Although the AViSSS scripting is needed to guide subjects through the paths in an intentional, ordered fashion, giving them some freedom to explore environments would be beneficial. This would increase their enjoyment of the experience and perhaps allow greater generalization of the scenarios. This will require applying rule-based scripts to each virtual character to allow for appropriate responses. It will also require controls for interacting with the environments.

Another possible direction for future development is to use alternative input devices (such as a Wii controller). In addition, eye tracking would be advantageous in teaching individuals with AS good eye contact, which frequently proves a challenge.

References

1. P. Mitchell, S. Parsons, and A. Leonard, "Using Virtual Environments for Teaching Social Understanding to 6 Adolescents with Autistic Spectrum Disorders," *J. Autism and Developmental Disorders*, vol. 37, no. 3, 2007, pp. 589–600.
2. S.E. Gutstein and T. Whitney, "Asperger Syndrome and the Development of Social Competence," *Focus on Autism and Other Developmental Disabilities*, vol. 17, no. 3, 2002, pp. 161–171.
3. G. Junker, *Pro OGRE 3D Programming*, Apress, 2006.
4. S. Burbeck, "Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC)," Univ. of Illinois at Urbana-Champaign (UIUC) Smalltalk Archive, 1992; http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

***Justin A. Ehrlich** is a doctoral student in the University of Kansas Department of Electrical Engineering & Computer Science. Contact him at jaehrlic@ku.edu.*

***James R. Miller** is a professor in the University of Kansas Department of Electrical Engineering & Computer Science. Contact him at miller@eecs.ku.edu.*

Contact editor Mike Potel at www.wildcrest.com.