THESIS

MULTIMODAL AGENTS FOR COOPERATIVE INTERACTION

Submitted by

Joseph J. Strout

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2020

ABSTRACT

MULTIMODAL AGENTS FOR COOPERATIVE INTERACTION

Embodied virtual agents offer the potential to interact with a computer in a more natural manner, similar to how we interact with other people. To reach this potential requires multimodal interaction, including both speech and gesture. This project builds on earlier work at Colorado State University and Brandeis University on just such a multimodal system, referred to as Diana. I designed and developed a new software architecture to directly address some of the difficulties of the earlier system, particularly with regard to asynchronous communication, e.g., interrupting the agent after it has begun to act. Various other enhancements were made to the agent systems, including the model itself, as well as speech recognition, speech synthesis, motor control, and gaze control. Further refactoring and new code were developed to achieve software engineering goals that are not outwardly visible, but no less important: decoupling, testability, improved networking, and independence from a particular agent model. This work, combined with the effort of others in the lab, has produced a "version 2" Diana system that is well positioned to serve the lab's research needs in the future. In addition, in order to pursue new research opportunities related to developmental and intervention science, a "Faelyn Fox" agent was developed. This is a different model, with a simplified cognitive architecture, and a system for defining an experimental protocol (for example, a toy-sorting task) based on Unity's visual state machine editor. This version too lays a solid foundation for future research.

ACKNOWLEDGEMENTS

I would like to thank my advisory committee for their thoughtful guidance and mentoring, and my family for their understanding and support over the last few years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Natural user interfaces are computer interfaces that allow users to interact with technology or virtual entities in ways that mimic how we interact with real-world entities. One type of natural user interface is the embodied conversational agent (ECA), which is presented visually with some form of embodiment, such as a human or animal figure, and which is able to engage the user in natural-language conversation.

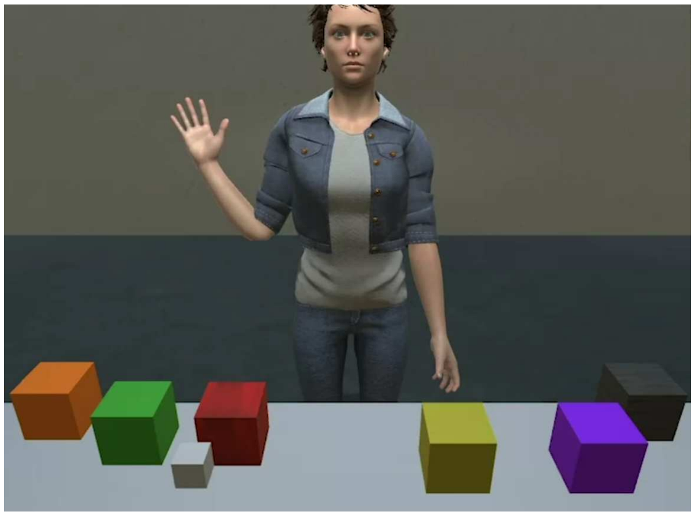


Figure 1.1: The original (version 1) Diana agent.

Over the last several years, an ECA referred to as Diana (Figure 1.1) [1] has been under active development. Diana is the result of a collaboration between two research groups and two universities. At Brandeis University, a team led by James Pustejovsky and Nikhil Krishnaswamy focused on natural language processing and reasoning, including 3D spatial planning and affordances. At Colorado State University, a team led by Ross Beveridge and Bruce Draper (and recently by Francisco Ortega) has focused on visual perception. Other aspects of the system, such as animation and inverse kinematics, have been worked on by both teams.

In the standard configuration, Diana appears as a life-sized human figure displayed on a large screen behind a work table. Diana is able to perceive and interpret the user's position, gaze direction, and a variety of gestures, including pointing, via a Kinect sensor and deep neural networks. Diana is also able to understand spoken speech, using off-the-shelf text-to-speech (TTS) services and the VoxSim reasoning system developed at Brandeis. On the output side, Diana is able to speak, gesture, and manipulate virtual blocks which appear on "her" side of the table. Diana has successfully demonstrated utility in cooperative tasks, where the agent and a human user work together using the shared perceptual space of the virtual blocks to build structures such as lines, pyramids, or staircases.

ECAs have been shown to be useful in a wide variety of contexts, particularly in education. ECAs have been found to facilitate learning by more actively engaging users' attention, and by making use of redundant communication channels (e.g., gesture as well as speech). In memory tests, recall was found to be higher with ECAs than without embodiment [2]; other work has found ECAs to be effective in teaching foreign languages [3] and math [4]. Some studies have found improvement of a full letter grade or more in student performance [5], comparable to results obtained by human tutors.

Other work has highlighted the benefits of ECAs for working with children with autism [6] [7] or cerebral palsy [8]. At Colorado State University, Lisa Daunhauer in the Department of Human Development and Family Studies is particularly interested the Down syndrome (DS) population [9] [10] [11]; and Anita Bundy in the Department of Occupational Therapy considers children with Developmental Coordination Disorder (DCD) [12] [13]. In all such populations there is an urgent need for precise, semi-automated measures to help assess impairments and developmental delays, both to identify children who may benefit from early intervention, and to evaluate outcomes in clinical trials. This has led to a new collaboration between these researchers and Ross Beveridge and Francisco Ortega in the Department of Computer Science, to explore applications of an advanced ECA like Diana in the context of developmental outcomes.

However, one limitation of early versions of the Diana system was that it could not be smoothly interrupted if, for example, it grabbed the wrong block. To avoid such a situation, Diana was therefore programmed to ask the user for confirmation before almost any action, but this turned out to be a slow and unpleasant user experience as well. Fixing this and related issues was proving difficult due to fundamental limitations of the system architecture.

To address this problem, a new system architecture was developed with a clean-sheet design. This has resulted in a new, “Diana 2.0” system that is more responsive and more proactive. It can be interrupted, and so has less need for confirmation before acting, resulting in a system that is faster and easier to use. The new architecture is also more modular, making it easier to produce variants of the system with different feature sets.

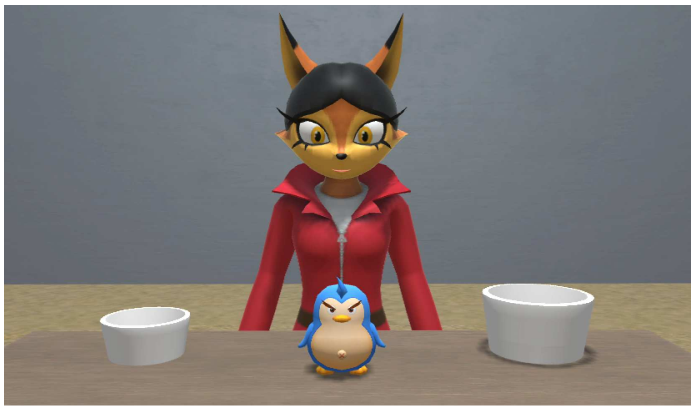


Figure 1.2: The new Faelyn Fox avatar, illustrated here in the context of the first user test.

One example of such a variant is the Faelyn Fox system, an ECA specialized for the new developmental outcomes work (see Figure 1.2). In addition to a different appearance and voice, Faelyn Fox uses a simplified cognitive architecture and a visually-edited behavior system, allowing non-programmers to develop responsive yet clearly controlled experimental protocols. It is expected that this ECA will find use in assessment tasks in young children. (Both the new Diana 2.0 system architecture, and the Faelyn Fox system, are primarily the work of this author.)

The rest of this thesis is organized as follows. Chapter 2 reviews related research on virtual worlds, embodied agents, and child development, providing a historical context for the Diana and

Faelyn systems, and laying the foundation for future work in developmental assessment. Chapter 3 describes my contributions to Diana 2.0, highlighting, in particular, the blackboard architecture which accounts for many of the new system's advantages. Chapter 4 presents Faelyn Fox and the new software modules built specifically for that ECA, including the visual behavior editor. Chapters 3 and 4 together may serve as useful reference material for future maintenance and development of the software. Finally, Chapter 5 considers possible directions for future research and attempts to place the project in a larger historical context.

Chapter 2

Background

2.1 Natural user interfaces

Virtual worlds and embodied agents have a rich history in computer science. In many cases a key feature of such agents is a gestural interface, often in combination with spoken or written language. Such interfaces involve fusing different types of data, as well as managing dialog state over time [14]. Early work in the field studied the gestures used in collaborative tasks, dividing them into *pointing* and *representational* gestures [15]. Later work on iconic gestures (those in which the path or pose of a gesture depicts what is described) found that the content of these gestures cannot be understood without reference to the shared context of the speaker and listener [16].

However, Donald Norman [17] pointed out that gestural interfaces (often referred to under the marketing term "natural user interfaces") are not natural at all; that is, gestures are usually culture-specific, not discoverable, and offer little feedback, especially when compared to graphical user interfaces. This highlights the value of a multimodal interface, where gestures and speech can be used to disambiguate each other. Kennington et al. [18] described a model for interpreting a speaker's intentions considering both spoken language and nonverbal cues such as gestures and gaze, as well as a method for grounding those representations via the shared visual context. It extends previous work on grounded semantics by working incrementally (i.e. word by word).

2.2 A Brief history of ECAs

A review of earlier work in ECAs helps put Diana and Faelyn Fox into the broader context of the field. A selection of ECAs appears in Figure 2.1. An early ECA named August (Figure 2.1(a)) served as a museum guide in Sweden [19]. The authors collected information on how naive users responded to the system and how they adapted their speech when the agent failed to understand.

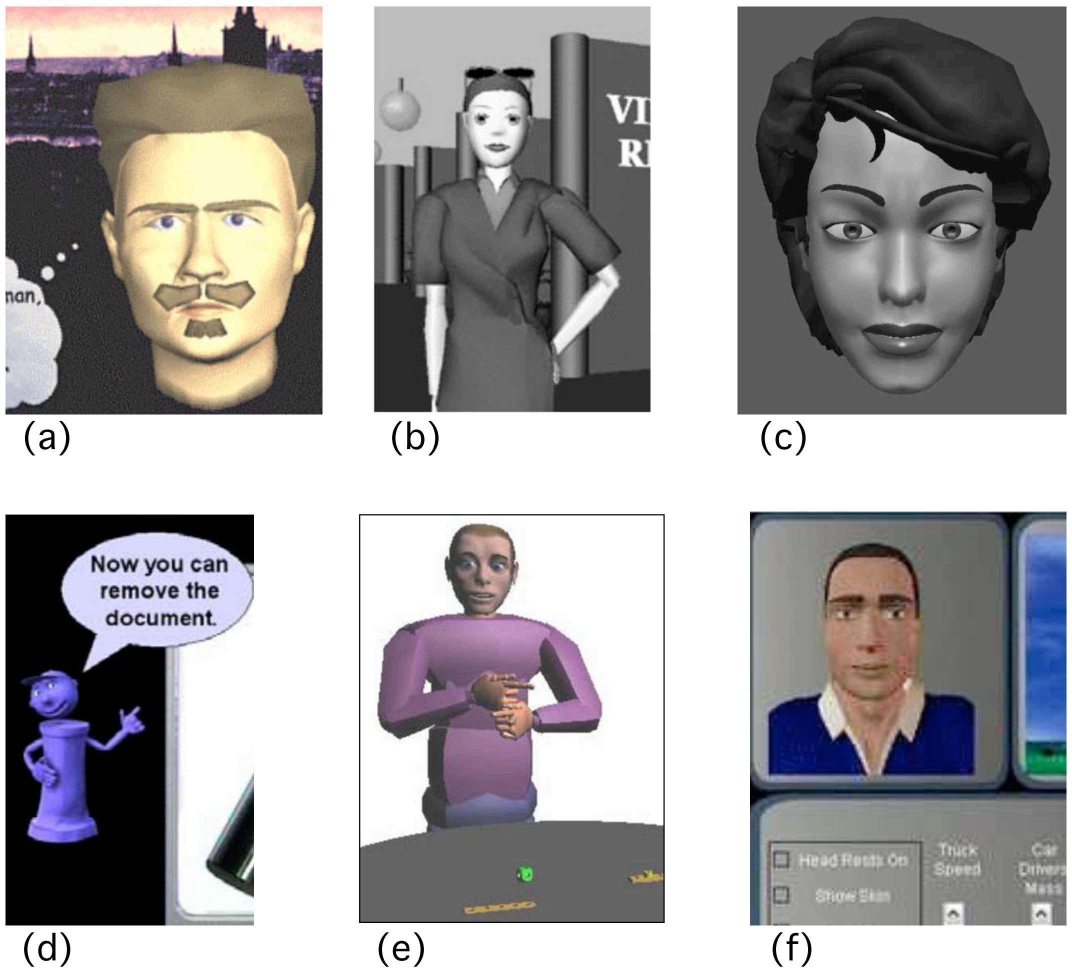


Figure 2.1: A chronological survey of ECAs in the literature. (a) August [19], 1999. (b) Rea [20], 2001. (c) Greta [21], 2002. (d) SmartKom [22], 2003. (e) Max [23], 2003. (f) AutoTutor-3D [24], 2014. See text for details.

Cassel et al [20] studied the nonverbal cues associated with topic shifts in discourse, and incorporated these results into Rea, an ECA real-estate agent that communicated through speech and gesture (Figure 2.1(b)). The following year, Pelachaud et al [21] presented Greta (Figure 2.1(c)), an ECA for discussing medical information. Greta featured a belief-desire-intention network for simulating emotional states.

SmartKom was a symmetric multimodal agent (i.e., one in which all modalities are available for both input and output) that uses speech, gesture, and facial expression [22] (Figure 2.1(d)). SmartKom used a three-tiered representation of discourse, with layers for domain, discourse, and modality, and also understood pointing gestures from the user.

Kopp et al [23] presented Max, a 3D (CAVE-based) virtual agent who worked with a human user on assembly tasks on a virtual table, in a setup is very similar to our Diana system (Figure 2.1(e)). Max is built on MURML, an XML-based language to represent communication acts that combine speech, gesture, and facial behaviors, and a software system for smoothly linking these together [25]. Like August, Max was also put to work as a museum guide [26], where analysis of conversation logs found that museum goers were likely to use human-like communication strategies with this ECA in a real-world setting; in this case he was displayed at human-like size on a static screen, standing face-to-face with museum goers, and equipped with camera-based visual perception.

All the ECAs discussed so far were built between 1999 and 2003. See [27] for a review of major projects and progress in ECAs in the first decade of the 2000s, including the realization of both verbal and non-verbal behaviors. Somewhat newer is AutoTutor and its several variants, such as AutoTutor-3D [24], a multimodal ECA that analyzes students' speech acts and adapts accordingly (Figure 2.1(f)).

The Diana system builds upon this previous work in important ways. Like Max, but unlike the other ECAs mentioned above, Diana uses a full-sized, full-body, humanlike avatar, with a shared perceptual space representing realistic objects. But Diana's multimodal communication is grounded in human-human elicitation studies on the same blocks world building task [28]. Researchers found that tasks were completed faster with gesture and speech together than with speech alone. They also found that social cues (e.g., acknowledgement) are frequently used between humans, and point out that these are often omitted from artificial gesture sets. Gestures are grouped into three categories: social, deictic, and iconic; iconic gestures were found to be less important when speech was allowed. They also found that when ambiguities arise, the conversational lead switches to the other partner in order to resolve the ambiguity.

Using VoxML as its cognitive foundation [1], Diana was built to emulate these communication strategies, leading to further studies exploring peer-to-peer communication between people and virtual agents [29].

2.3 Benefits of ECAs

The efficacy and benefits of ECAs over non-embodied agents or other types of user interfaces have been the subject of much research. It is generally recognized that ECAs engage users more actively, and by being a social presence, may increase motivation. It is also pointed out that ECAs enable the use of multiple communication channels, allowing for a failure in one channel to be recovered by another, or a message in one channel to be explained or elaborated by another [2]. A review of evidence supporting and disputing the benefit of ECAs on learning and memory was presented by Louwerse et al [30], who provided evidence from eye tracking that humans treat ECAs as conversational partners, not mere attention magnets.

Bickmore et al [31] performed two large clinical trials showing that patients respond well to an ECA providing health information, and found this especially beneficial for patients with inadequate health literacy. Comments from participants suggested that even patients who do not typically like to interact with computers liked interacting with the ECA.

Beun et al [2] tested memory on stories that were presented by a realistic human female virtual agent, a purple cartoon gorilla, or by no ECA at all. Recall was found to be higher with both virtual agents than with the absent ECA, but anthropomorphism (measured to be higher for the humanlike agent) had no apparent effect. This suggests that ECAs are effective, but not because of anthropomorphism, and has obvious relevance to our Faelyn Fox agent.

Matus et al [32] presented a machine learning system that learns to understand deictic gestures in which the speaker is drawing attention to specific objects, and found that combining multiple modalities of communication (speech and gesture) is more effective than using only a single type. The integrated system uses a robot arm to manipulate colored blocks from unscripted directions of human users, similar to our objectives with the Diana blocks world setup.

A study by Andrist et al [33] shows the effectiveness of *bidirectional gaze*, i.e. the coordinated production and detection of gaze cues in a collaborative task. They find that this form of nonverbal communication with a virtual character can be established using eye-tracking glasses or (more simply) head-pose estimation, and that it works both for agents on a flat screen and for agents in VR. A similar effect was found in a study involving an ASIMO robot telling a story to human listeners [34]; it found that listeners were significantly better at recalling story details later when the robot had looked at them more.

In an educational context, ECAs have been shown to be effective for learners of foreign languages [3] and math problems [4]. Kumar and Rose [5] present an architecture for building conversational agents that work in a collaborative-learning (i.e. multi-user) environment. They report findings of improvement of a full letter grade or more in comparison with no learning support, or static (non-conversational) support conditions.

Graesser et al [24] describe computer agents that simulate human tutoring, with an emphasis on AutoTutor. Empirical evidence suggests that AutoTutor and similar natural-language computer tutors produce learning gains comparable to those of trained human tutors, in a variety of subjects. AutoTutor has been made in various versions, including various combinations of communication modalities. In one direct comparison, AutoTutor-3D (see Figure 2.1(f)) yielded a significant improvement in learning over a strictly conversational version for students who used the system more than once. In another study, an AutoTutor that was emotionally empathetic aided learning better than one that was not emotionally responsive.

Some researchers have looked at the benefits of ECAs for users with disabilities. Mencía et al [8] relate experiences and recommendations pertaining to the use of ECAs with children who have severe motor and mental disabilities (primarily cerebral palsy). Specific recommendations included the need to clearly show the mouth and eyes, and to display exaggerated facial expressions along with both auditory and gestural reinforcements. Bosseler and Massaro [6] showed that an ECA was effective at teaching vocabulary to children with autism. Tanaka et al [7] found ECAs to also be effective at social skills training in people with autism spectrum disorders, and that the multimodal nature of such ECAs was central their success.

It is worth noting that many users today have experience with non-embodied conversational agents that have become common home appliances, such as Amazon Echo and Google Home, as well as agents built into our smart phones. However, Luger and Sellen [35] find that user expectations are out of step with the reality of these conversational agents. These observations emphasize the importance of natural language as a communications channel, while at the same time highlighting how much work is still to be done to make such interfaces truly natural.

2.4 Developmental disorders and assessment

The incidence of Down syndrome (DS) has increased over 30% from 1979 through 2003 [36]. Mouse models of DS show that pharmacological interventions can rescue some of the intellectual disability resulting from DS, and that the benefits can outlast treatment. However, such interventions are more effective the earlier they are applied [37].

Developmental coordination disorder (DCD) is a neuromotor condition affecting about 5%-6% of school-aged children [38]. It is characterized by difficulty coordinating either gross or fine motor movements, or both. It is typically diagnosed at age 5 or later. Earlier diagnosis and intervention might offer better outcomes, and is an area of active research.

Various tasks involving toys have been successfully used for assessment of children of 3-4 years of age [39]. Standard tests include the Bayley Scales of Infant Development (BSID-IV) [40], which can assess children up to 42 months old, and the Peabody Developmental Motor Scales (PDMS-2) [41], which can be applied from birth through 5 years old.

Information processing abilities such as attention, processing speed, and memory in infants and toddlers predict later executive function [42]. Useful measures include reaction time and duration of engagement, both of which could be measured automatically while interacting with a system like Faelyn Fox.

Chapter 3

Contributions to Diana 2.0

The Diana 2.0 system was a clean-sheet design, with relatively little code retained from the original system outside of VoxSim itself. Like its predecessor, Diana 2.0 is written in C# within the Unity development environment.

3.1 Cognitive Architecture

The 2.0 cognitive architecture is built around a *blackboard* metaphor [43] [44]. Our version of the blackboard is a strongly-typed key-value store, implemented as a C# class called DataStore. DataStore follows a modified Singleton pattern [45], with a static instance that is commonly used in actual application, though other instances may also be created for such purposes as unit testing.

The keys in our DataStore are strings, with conventions that subdivide the key space into progressively more specific areas. For example, all keys that begin with "user:" refer to some information about the human user, while all keys that begin with "me:" reflect some state about the agent. Examples of keys currently in use are given in 3.1, and a full list may be found in the GitHub repo as DataStore.md.

Table 3.1: Sample keys used with the DataStore, along with the data type and a brief description.

Key	Data Type	Description
me:intent:action	String	action agent intends to do: "point", "grab", etc.
me:intent:target	Vector3	target location of 'me:intent:action'
me:name	String	agent's own name, e.g. "Diana" or "Faelyn Fox"
me:speech:intent	String	text the agent intends to speak
user:intent:location	Vector3	location to which the user wants the agent to attend
user:isPointing	Boolean	true iff the user is pointing
user:isSpeaking	Boolean	true iff the user is speaking
user:speech	String	text of utterance user has just spoken (or typed)

The values in the data store are small objects that implement a DataStore.IValue interface. This interface defines several methods useful for all types: ToString (string conversion), Equals (equality testing), and IsEmpty (used with DataStore.HasValue). Concrete classes implementing this interface were written for string, boolean, integer, scalar, float array, Vector3, and quaternion (orientation/rotation). Core DataStore methods allow any type to be set or retrieved by string key; convenience methods allow for setting and retrieving these specific datatypes without having to box or unbox the values on each call.

While users of the DataStore may poll it for values of interest, an event system was also made available to simplify the code and reduce the overhead of polling. Code may subscribe to data changes in two different ways. First, it may subscribe to changes to a specific key, with a callback invoked only when that key is changed. Alternatively, it may subscribe to an event that fires whenever any value is changed. These events receive both the key and the new value, enabling the subscriber to disambiguate the change when needed.

With this DataStore blackboard as the interface hub, a large number of relatively small, simple modules were written. Each of these works by subscribing to (or polling for) changes to keys of interest; applying some computation (often, updating a state machine); and then updating the blackboard with new values. In addition, of course, there are "input" modules which take values from some other source (e.g. speech transcribed by a Text-to-Speech system) and place them on the blackboard, and "output" modules which take blackboard values and use them to cause some other effect (e.g. setting inverse kinematics targets for the avatar).

These modules are supported by a simple common base class called, uncreatively, ModuleBase. ModuleBase provides some debugging support: a comment that is stored whenever one of its value setter methods is used, and a link to a ModuleDisplay, which is a UI component that displays module state (including these comments). Particularly in the early days of Diana 2.0, these module displays were used extensively to provide a continuous view into the state of the system.

At the time of this writing, nearly 50 ModuleBase subclasses have been created. Table 3.2 lists the modules created by this author.

Table 3.2: Cognitive modules (subclasses of ModuleBase) written primarily by the author.

Name	Description
AttentionModule	state machine controlling agent's attention and alertness
BoolKeyStandInModule	allows boolean keys to be set by holding a keyboard key
CommandsModule	part of reduced NLP system; interprets commands from parsed text
DianaGaze	makes a humanoid agent look at a target by turning head and eyes
EyeControlModule	controls eyes, including gaze, alertness display, and blinking
GrabPlaceModule	uses IK to make a humanoid agent pick up or place an object
InteractionGlue	interfaces between blackboard intents and hand interaction system
ParserModule	part of reduced NLP system; interprets raw text from the user
PleasantriesModule	part of reduced NLP system; handles phatic communication
PointAtPointModule	produces behavior of pointing wherever the user points
PointModule	uses IK to make a humanoid agent point at any target location
SelfKnowledgeModule	provides the agent with knowledge of its name, voice, and hands
SocketInterfaceModule	provides a network interface to the DataStore
SpeechOutModule	produces audible speech using some text-to-speech service
SRStandInModule	allows a text input field to substitute for speech-to-text
StanfordParserModule	uses StanfordNLP library to convert text into a parse tree
StringKeyStandInModule	allows string keys to be set by pressing certain keys on the keyboard
WatsonStreamingSR	does streaming text-to-speech via the IBM Watson service

3.2 Network Interface

For some functions — most notably deep neural network models built on PyTorch — it was necessary to interface with external processes, which in some cases might not even be running on the same machine.

To facilitate this, a SocketInterfaceModule was written in C# which acts as a simple server, listening for incoming TCP/IP connections on a given port (by default, 38276). The module services the socket on a worker thread to avoid blocking the main thread (where all Unity classes do their regular updates).

The code in this module implements a command protocol, based loosely on the open-source Redis data store [46], as shown in table 3.3. When a client process uses the SUB command to subscribe to value changes for a particular key, it will receive SETI, SETS, SETV, or SETB messages using the same syntax.

To demonstrate how to interface with the system from Python, a simple client script called pythonSockDemo.py was written (available in the GitHub repo). This establishes a TCP/IP connection with the SocketInterfaceModule running in Unity, and provides helper methods for setting values or subscribing to a particular key. As a further demo, it increments a counter once per second and puts its value on the blackboard as "demo:python:counter". Based on this example, other members of the lab were able to interface other Python code to the Diana 2.0 system for such things as gesture recognition.

Table 3.3: Network protocol understood by the SocketInterfaceModule.

Command	Example	Meaning
SETI <key> <integer value>	SETI me:alertness 90	set an integer value
SETS <key> <string value>	SETS user:speech Hello	set a string value
SETV <key> <x> <y> <z>	SETV user:pointPos 2.5 1.7 0	set a Vector3 value
SETB <key> <T or F>	SETB user:isPointing T	set a boolean value
SUB <key>	SUB me:isEngaged	subscribe to value changes
NAME <new name>	NAME HandClient	changes client name

3.3 Agent Independence

One goal in the new design was to make the agent readily swappable. In fact, the new system was built with not one, but four different agents: the original Diana model, the new Diana model, a humanoid robot, and an industrial robot arm. See Figure 3.1.

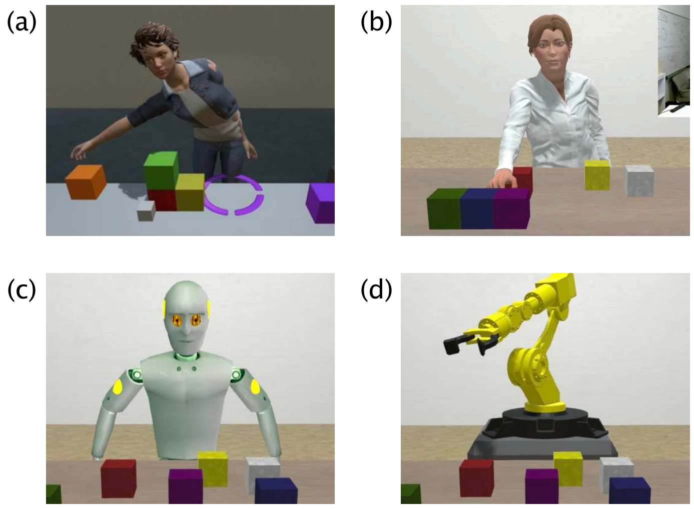


Figure 3.1: Four agents functional in the early days of Diana 2.0. (a) The original Diana model. (b) The new Diana model. (c) A humanoid robot. (d) An industrial robot arm.

The new Diana model was built from an Adobe Fuse character configured by the author and rigged (i.e. given a standard humanoid animation skeleton) through the Mixamo service. It features 67 degrees of freedom, including fully articulated hands.

Currently only the new Diana model is in use, plus Faelyn Fox (described in the next chapter). However, for a while in the early days of the project, all four agents were functional. The original Diana model has since been deleted from the project, as it was based on a third-party asset that is no longer supported and does not work with recent versions of Unity. The two robotic agents, however, remain in the project and could be made to work again with relatively little effort.

This is accomplished primarily by taking advantage of the decoupling offered by the DataStore (blackboard). Each agent includes a SelfKnowledge module that places the agent's name, hand bones, and voice options onto the blackboard. Other modules, which are placed not on the avatar but in a general "cognitive architecture" part of the scene, pick up these values and use them appropriately. Modules which directly control some aspect of the model, such as eye/eyelid control or other motor control, are placed on each agent. All the humanoid agents use identical scripts for controlling reach and hand pose, but they use different scripts for eye control and facial expressions, since (for example) the humanoid robot has considerably fewer degrees of freedom in its face. The robot arm uses a different IK setup for reaching, and of course has no face at all. Other cognitive modules continue to post face-related intents to the blackboard, and these are harmlessly ignored in this case.

3.4 Inverse Kinematics

Like the Diana 1.0 system, Diana 2.0 relies largely on inverse kinematics (IK) to control movement of the hands, torso, neck, and head. We are using a commercial library (FinalIK) to do the actual IK computations. However, proper setup and configuration is essential to avoid problems such as reaching through the body, bending awkwardly at the waist, or looking away while reaching for an object. This section provides some details on how all this was set up to produce Diana 2.0's smooth, natural-looking motion.

Three FinalIK components are added to the agent model: Full Body Biped IK, Look At IK, and Interaction System. Full Body Biped IK is the primary IK controller. It requires references to joint transforms throughout the model skeleton. It also has detailed parameter blocks for the body, each arm, and each leg. For our purposes, the Left Arm and Right Arm parameter blocks are particularly important; these are involved whenever the agent reaches for something on the table, and by default, will often result in solutions that have the agent's arms awkwardly close to her sides, even while successfully reaching a target location on the table.

The solution to this problem was to add a "Bend Goal" transform to each arm, or at least to the right arm (since currently our agent is only programmed to reach with her right hand). This is a transform positioned in the environment about 0.8 meters to the side, and slightly (27 cm) behind the agent, and then set as the Bend Goal in the Right Arm parameter block, with a Bend Goal

Weight of 0.2. This additional constraint for the IK solver causes the agent to lift her elbow away from the body, all other things being equal, resulting in a much more natural reaching motion.

The other important setting to avoid awkward reaches is the "Reach" parameter under "Chain" (still in the Right Arm parameter block). Larger values here will cause the upper body to bend more aggressively toward the reach goal. A value of 0.1 is appropriate, producing a gentle bend towards the reach target.

The Look At IK component is used to make the agent turn her head towards a target. This must be used whenever reaching is used, or the IK solution will cause the head to turn away from the reach position as a side-effect. Note that we are not using FinalIK to control the eyes themselves, as every model seems to have the eyes set up differently. A custom script, DianaGaze.cs, was written to both rotate the eyes to look ta a target, and control the Look At IK component to turn the head.

Finally, the Interaction System component is used to provide fine control over reaching and grasping motions. It works in conjunction with an Interaction Object script on any graspable object, that defines the hand pose (or poses) used to hold that object. Our use of these components is standard, following the FinalIK instructional videos. To interface this with our custom cognitive architecture, an InteractionGlue.cs script was created; this reads several DataStore keys and invokes methods on Interaction System accordingly, and also sets a key when the intended action is complete.

Because they operate directly on IK components attached to a specific agent, both DianaGaze and InteractionGlue are attached directly to the agent, rather than grouped with the agent-independent scripts at the scene level.

3.5 Facial Control

In addition to gross body movements, there is a need for fine control over the agent's facial expressions, eye and eyelid movements, and mouth movements (lip syncing). These are handled with several new scripts.

As noted in the previous section, a DianaGaze.cs script controls the direction of the eyes, using the corresponding bones in the skeletal system. These are hooked into the animation system via Unity's standard OnAnimatorIK callback, so after the eyes are positioned by any running animation on each frame, we override their rotation to make them look at the target position.

All other aspects of facial control are done not through bones, but through blend shapes. A blend shape is a modification of a 3D mesh, represented as the change in position for each vertex in some subset of the mesh vertices. These deltas are multiplied by a weight for each blend shape, and added to the normal (default) position of each vertex to determine the final shape of the mesh. As Diana 2.0 uses an Adobe Fuse model, we have access to the 50 blend shapes defined on all Fuse models.

A script called EyelidMorphs.cs was written to read the key me:eyes:open on the blackboard, and set the weight of the Blink_Left and Blink_Right blend shapes accordingly. Thanks to other modules in the cognitive architecture, this results in Diana blinking, closing her eyelids slightly when bored, and opening them wider than usual when excited (or more accurately, when the intent is to produce the illusion of boredom or excitement).

While the appropriate weights for closing the eyes are simple, facial expressions are more complex, requiring the adjustment of 5-10 different blend shapes such as Brows_In_Left and Mouth-Narrow_Right. To facilitate this, a BlendShapeMixer.cs script was written. This allows one to define any number of "expressions" as a combination of any number of blend shapes. Other code (e.g. FaceUpdate.cs, written by colleague Heting Wang) then adjusts the weights on these expressions, working at the higher abstraction level of expressions rather than at the lower level of individual blend shapes. This is used to express emotion, to study the role of affect between the ECA and the human user [47].

3.6 Natural Language Processing

Since VoxSim was reintegrated into Diana 2.0, it has handled all the agent's natural language processing (NLP) needs. However, prior to that point, a simpler NLP system was implemented, which served to get the new system up and running with a conversational interface quickly. This section describes the design and operation of that NLP system.

Processing begins by dividing the input into words, and tagging each word with its most likely part of speech. The tags used are shown in table 3.4, and are a subset of tags used by the Brown corpus [48]. To assign these tags, a dictionary was built by analyzing the SEMCOR 3.0 data set [49] as follows. For every word (and set phrase), I collected all the instances in the corpus and counted its usage as various parts of speech (POS). Then I output a simple file with the word, and the POS frequencies, normalized to 1.

Table 3.4: Part of speech tags used by the NLP module.

Tag	Meaning
NN	noun
JJ	adjective
CD	ordinal number (one, two, etc.)
CC	conjunction (and, or)
IN	preposition
RB	adverb
DT	determinant
NP	proper noun
VB	verb
RP	adverb particle (off, up)
WRB	WH-adverb (where)
WDT	WH-determinant (also where)
WP	WH-pronoun (who, what)
PPS	singular pronoun (it, one)
UH	interjections and greetings

While the result does a good job of correctly identifying the POS of many words, it fails miserably on others. For example, “point” is overwhelmingly classified as a noun (97% of the time) if not capitalized, or a proper noun (100% of the time) when capitalized (even if at the start of a sentence). This is because the SEMCOR corpus consists almost entirely of declarative statements (newspaper articles etc.), with very few imperative commands or interrogative queries.

This is an ill fit for our purposes, as interaction with a computer agent consists almost entirely of commands and queries, with very little need for declarative statements. So given a command such as “Point at the green block,” any system trained on a standard corpus like SEMCOR is going to begin from the difficult position of misidentifying the verb.

As I was unable to find a tagged corpus consisting of interrogative and imperative sentences, I worked around the issue by hand-coding a number of “overrides” of the SEMCOR parts of speech for specific cases. These include all the common verbs in our context (e.g. “point”, “pick”), some common emphatic words that were too rare in the corpus for correct identification (“hi”, “bye”), and some set phrases that were simply missed (e.g. “to the right of”, “pick up”, “multiplied by”). In addition, some things identified by the corpus as set phrases should not be, for example “two times” (treating this as a set phrase makes it very hard to parse mathematical expressions), so those were removed. In all, 48 such corrections were applied.

The words and their initial POS assignments are moved into a data structure that also includes dependency information, that is, which words are subordinate to which others. The objective of the next step is to produce a shallow parse — one in which noun phrases, verb phrases, and prepositional phrases are identified, but not necessarily parsing the entire input into a tree with a single root. A Parser.cs script was written for this purpose. It repeatedly applies an ordered set of 17 heuristics, any of which can modify the part of speech of one or more words, or assign dependencies. These heuristics are things like “a determinant modifies the closest noun or pronoun” and “an adverb right before a verb modifies that verb.” When no more rules can be applied, parsing stops.

These heuristics were constructed by hand with the aid of extensive unit testing. Each rule, when applied, returns a unique rule ID, which greatly helps debugging. In addition, the parse result can convert itself into a string representation which includes both POS and dependency information, while still being easily read, enabling simple one-line test cases. The general philosophy behind the rules was to handle all the easy cases, while leaving difficult or ambiguous cases unresolved — possible since we are doing a shallow parse rather than a complete parse.

The next step of the NLP pipeline is to interpret the *intent* and *arguments* of the input. This is represented with a small class tree based on on a Communication base class. Each subclass represents a different communication type or intent, as shown in table 3.5. Each subclass has public fields for whatever arguments are appropriate for that type.

Filling out an appropriate Communication instance is the job of the Grok.cs script. This script too relies on hand-coded heuristics based on both part of speech and specific words, and supported by extensive unit tests. The main method invokes helper methods specialized for interpreting object references, locations/directions, and actions. The code is written in such a way that even if the entire input cannot be parsed, it is likely to find *some* bits of meaning, giving client code a chance to at least produce a helpful response.

Everything described in this section up to this point is completely encapsulated in a CWCNLP namespace, and independent of the DataStore, cognitive modules, or anything else in this particular project. Two cognitive modules tie it into the rest of the system. First, ParserModule.cs takes user input from the blackboard (via a user:speech key), runs it through the parser, and posts the final Communication (using a user:communication key). And finally, a CommandsModule subscribes to the latter key, and takes action based on the various communication types and arguments.

As noted in the beginning of this section, the NLP architecture described here is no longer in use; modern Diana 2.0 uses VoxSim instead. However, for a while, this NLP system was used successfully to handle inputs such as those in table 3.6.

3.7 Support Code

In the last section of this chapter, I cover various support bits of support code to make maintenance and debugging easier. These including unit testing, debug displays, and runtime options. Some of these features were previously mentioned, but more detail is presented here about how they actually work.

Unit testing is supported by a custom UnitTest script, which defines virtual methods Setup, Run, and Cleanup. Subclasses of this are written to test various areas of functionality in the project;

Table 3.5: Classes used to represent communication from the user in the NLP system.

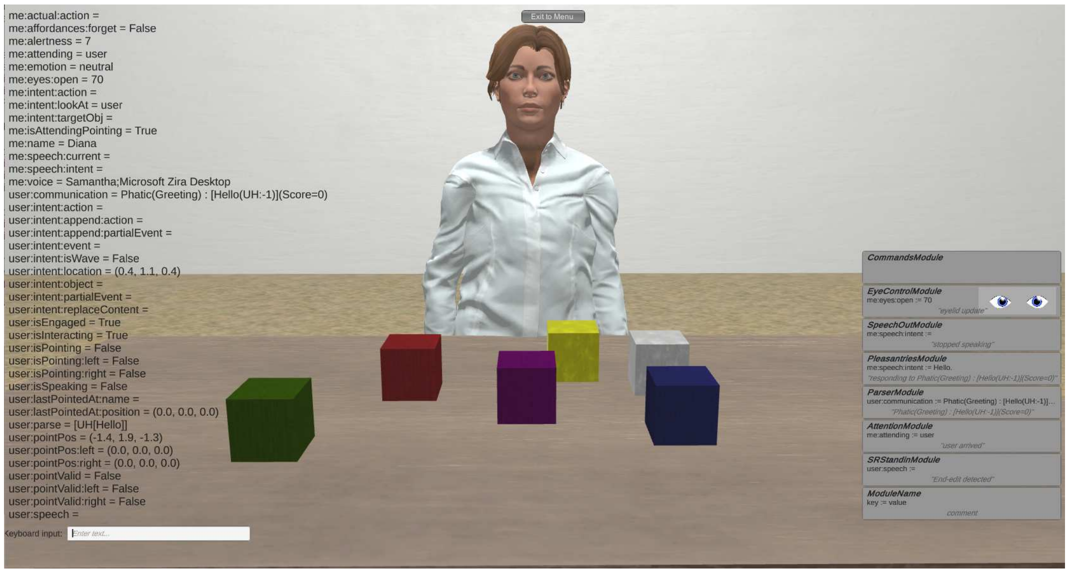
Class	Meaning
ComCommand	commands
ComQuery	questions
ComAssertion	imparting information to the listener
ComConfirmation	agreement/confirmation
ComDenial	disagreement/denial
ComEmote	interjection or expression of emotion/state (wow, argh, etc.)
ComPhatic	back-channel communication; conversational grease

Table 3.6: Examples of user input correctly interpreted and handled by the custom NLP system.

Hello!
What is your name?
Pick up this block.
Lift the green block.
Grab that one.
Put it down.
Place this one here.
Put it over there.
That’s enough.
Stand by.
What is this block?
Good-bye.

these override at least the Run method. The base class also defines a number of assertion functions for comparing actual values to expected values. Failures are reported through Unity's standard Debug.LogError mechanism.

The UnitTest subclasses are instantiated and run from a helper class called UnitTestRunner, which in the main (Demo) scene, exists on an otherwise empty GameObject of the same name, located under System in the scene hierarchy. Tests are run from the Start method, and so execute shortly after the scene is loaded.



```
me:actual:action =
me:affordances:forget = False
me:alertness = 7
me:attending = user
me:emotion = neutral
me:eyes:open = 70
me:intent:action =
me:intent:lookAt = user
me:intent:targetObj =
me:isAttendingPointing = True
me:name = Diana
me:speech:current =
me:speech:intent =
me:voice = Samantha;Microsoft Zira Desktop
user:communication = Phatic(Greeting) : [Hello(UH:-1)](Score=0)
user:intent:action =
user:intent:append:action =
user:intent:append:partialEvent =
user:intent:event =
user:intent:isWave = False
user:intent:location = (0.4, 1.1, 0.4)
user:intent:object =
user:intent:partialEvent =
user:intent:replaceContent =
user:isEngaged = True
user:isInteracting = True
user:isPointing = False
user:isPointing:left = False
user:isPointing:right = False
user:isSpeaking = False
user:lastPointedAt:name =
user:lastPointedAt:position = (0.0, 0.0, 0.0)
user:parse = [UH[Hello]]
user:pointPos = (-1.4, 1.9, -1.3)
user:pointPos:left = (0.0, 0.0, 0.0)
user:pointPos:right = (0.0, 0.0, 0.0)
user:pointValid = False
user:pointValid:left = False
user:pointValid:right = False
user:speech =
```

```
CommandsModule

EyeControlModule
me:eyes:open := 70
"eyelid update"

SpeechOutModule
me:speech:intent :=
"stopped speaking"

PleasantriesModule
me:speech:intent := Hello.
"responding to Phatic(Greeting) : [Hello(UH:-1)](Score=0)"

ParserModule
user:communication := Phatic(Greeting) : [Hello(UH:-1)]...
"Phatic(Greeting) : [Hello(UH:-1)](Score=0)"

AttentionModule
me:attending := user
"user arrived"

SRStandinModule
user:speech :=
"End-edit detected"

ModuleName
key := value
comment
```

Figure 3.2: Screen shot of the Diana 2.0 system, with blackboard state display (text on left side of image) and module displays (gray informational boxes at right) activated.

While the system is running, it is often useful to be able to monitor the state of various cognitive modules in real time. This is facilitated through *module displays* as shown in Figure 3.2 (right). Each display is connected to one cognitive module, and shows at least the most recently set DataStore key and value, along with the comment that was submitted with the value change. In some cases additional state information is shown, such as the EyeControlModule, which shows (via a cartoon-level diagram) the intended state of the eyes, including look angle and eyelid levels. This information remains valid and useful even when working with agents with limited or no eye articulation (such as the humanoid robot or industrial robot arm).

Also available is a filtered display of the current keys and values in the DataStore. Through Unity's inspector, this can be adjusted to show only keys which match some pattern (e.g "user:*" to show only user keys), or to hide any keys which match some pattern; by default it is configured to hide keys starting with "user:joint:*" which reflect the angles of every joint in the Kinect skeleton. An example of the DataStore display is shown in Figure 3.2 (left).

Finally, while these displays can be useful for debugging, they are unwanted clutter during normal operations. A user interface was therefore created that allows the operator to toggle these displays in any combination, along with other bits of stand-in UI (such as a speech stand-in that allows you to type your inputs rather than relying an speech-to-text). This options panel is summoned or dismissed by pressing the Alt key (or Option key on Mac), and all settings are stored in preferences so that they are retained across restarts of the system.

Chapter 4

Faelyn Fox

As noted in chapter 1, embodied agents have particular benefits when working with children. To maximize these benefits, and in consultation with Drs. Lisa Daunhauer and Anita Bundy, a new avatar was designed specifically for this context. The new agent, called Faelyn Fox, features both a new, more child-friendly model (see Figure 1.2), and a simplified cognitive model with some support for visually scripted experimental protocols.

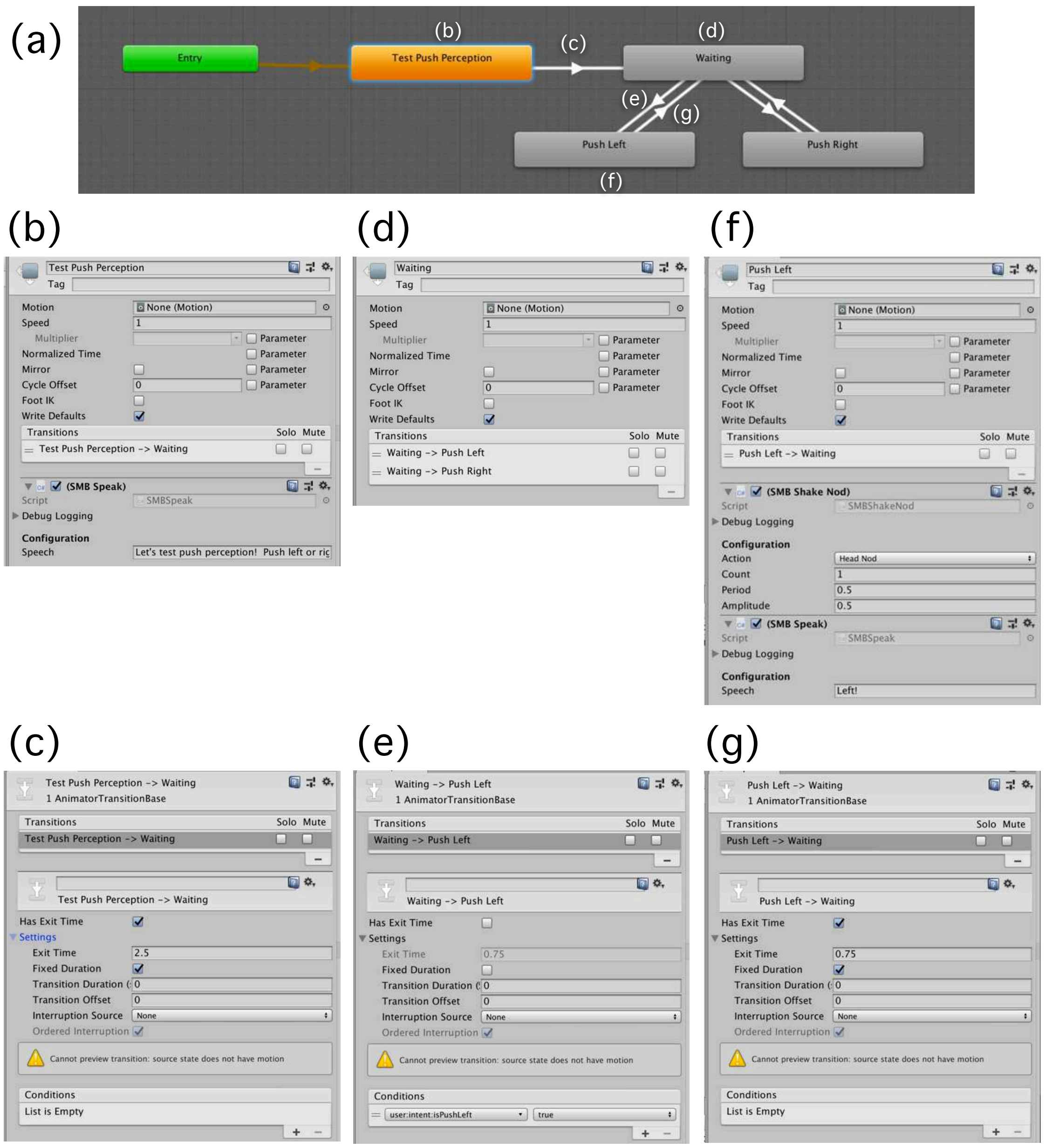
4.1 Visual protocol editor

The intended use of this agent differs from that of the Diana system in two important ways, and these differences have led to design changes "under the hood." First, children are less likely to effectively gage an artificial agent in conversation, and are certainly not likely to use complex linguistic constructs. Second, Faelyn Fox will be used to conduct test procedures which need to be highly repeatable, following a script that must be understandable by researchers in developmental science, rather than only by computer scientists.

As a result, the VoxSim cognitive architecture has been disabled in the Faelyn Fox agent. Currently, the agent has no language understanding at all, though basic understanding could be restored via the simplified parser and NLP modules described in section 3.6. To control the agent's behavior, a visual scripting system has been developed, built upon Unity's standard animation state machine editor.

A small example of such a protocol state machine is shown in figure 4.1. This particular state machine implements a simple protocol for testing the agent's perception of push left/right gestures from the user: the agent speaks a greeting, waits for a gesture, identifies the gesture seen, and returns to the waiting state.

Unity's state machine editor is normally used for controlling animations; the state machine configuration is stored in a type of asset called an AnimationController. Here, we are using it to



(b)

Test Push Perception
Tag
Motion: None (Motion)
Speed: 1
Multiplier: ☐ Parameter
Normalized Time: ☐ Parameter
Mirror: ☐ ☐ Parameter
Cycle Offset: 0 ☐ Parameter
Foot IK: ☐
Write Defaults: ✓

Transitions	Solo	Mute
Test Push Perception -> Waiting	☐	☐

✓ (SMB Speak)
Script: SMBSpeak
Debug Logging
Configuration
Speech: Let's test push perception! Push left or ric

(d)

Waiting
Tag
Motion: None (Motion)
Speed: 1
Multiplier: ☐ Parameter
Normalized Time: ☐ Parameter
Mirror: ☐ ☐ Parameter
Cycle Offset: 0 ☐ Parameter
Foot IK: ☐
Write Defaults: ✓

Transitions	Solo	Mute
Waiting -> Push Left	☐	☐
Waiting -> Push Right	☐	☐

(f)

Push Left
Tag
Motion: None (Motion)
Speed: 1
Multiplier: ☐ Parameter
Normalized Time: ☐ Parameter
Mirror: ☐ ☐ Parameter
Cycle Offset: 0 ☐ Parameter
Foot IK: ☐
Write Defaults: ✓

Transitions	Solo	Mute
Push Left -> Waiting	☐	☐

✓ (SMB Shake Nod)
Script: SMBShakeNod
Debug Logging
Configuration
Action: Head Nod
Count: 1
Period: 0.5
Amplitude: 0.5
✓ (SMB Speak)
Script: SMBSpeak
Debug Logging
Configuration
Speech: Left!

(c)

Test Push Perception -> Waiting
1 AnimatorTransitionBase

Transitions	Solo	Mute
Test Push Perception -> Waiting	☐	☐

Test Push Perception -> Waiting
Has Exit Time: ✓
Settings
Exit Time: 2.5
Fixed Duration: ✓
Transition Duration (: 0
Transition Offset: 0
Interruption Source: None
Ordered Interruption: ✓
Cannot preview transition: source state does not have motion
Conditions
List is Empty

(e)

Waiting -> Push Left
1 AnimatorTransitionBase

Transitions	Solo	Mute
Waiting -> Push Left	☐	☐

Waiting -> Push Left
Has Exit Time: ☐
Settings
Exit Time: 0.75
Fixed Duration: ☐
Transition Duration (: 0
Transition Offset: 0
Interruption Source: None
Ordered Interruption: ✓
Cannot preview transition: source state does not have motion
Conditions
user:intent:isPushLeft
true

(g)

Push Left -> Waiting
1 AnimatorTransitionBase

Transitions	Solo	Mute
Push Left -> Waiting	☐	☐

Push Left -> Waiting
Has Exit Time: ✓
Settings
Exit Time: 0.75
Fixed Duration: ✓
Transition Duration (: 0
Transition Offset: 0
Interruption Source: None
Ordered Interruption: ✓
Cannot preview transition: source state does not have motion
Conditions
List is Empty

Figure 4.1: Protocol state machine example implementing an interactive test of the agent's visual perception of push gestures. (a) State machine as viewed in Unity's animation controller editor; the initial state is indicated in orange. Each box represents a state, and arrows indicate transitions. (b) Inspector properties for the initial state. A *SMB Speak* state machine behavior has been added, causing the agent to speak the indicated text when this state is entered. (c) Inspector details for the transition from the initial state to the Waiting state. This transition has no conditions, but instead happens automatically after the Exit Time (2.5 seconds). (d) Details for the Waiting state. This state has no associated behaviors. (e) Details for the transition from Waiting to the Push Left state. This transition has a condition: it happens only when user:intent:isPushLeft on the blackboard becomes true. (f) Details for the Push Left state. Two custom behaviors have been attached: *SMB Shake Nod* has been configured to nod the agent's head, and *SMB Speak* provides verbal feedback. (g) Details for the transition from Push Left back to the Waiting state. With no conditions, this transition happens automatically after the Exit Time of 0.75 seconds.

instead control the agent at a higher level, where each state represents a step in some experimental procedure. This is possible due to several custom scripts that interface the state machine to the rest of the cognitive architecture. First is the *BlackboardToAnimParams* script. This continually sets any parameters defined on the animation controller from blackboard keys of the same name. For example, a boolean “user:isEngaged” parameter on the animation controller gets set by this script to the value of “user:isEngaged” on the blackboard on every frame. This allows state machine transitions to respond to blackboard values, as in 4.1(e).

The next piece is a set of simple scripts that cause behaviors in the agent (or the simulated environment) when certain states are entered. These are built on the *SMBBase* script, which itself derives from Unity’s *StateMachineBehaviour* class. Such classes can be attached to any state of an animation controller, and execute code when that state is entered, exited, or maintained, and any public fields may be configured for each such place they are attached. *SMBBase* provides utility methods to set values of various types on the blackboard. The *SMBSpeak* subclass uses these to set the “me:speech:intent” key with a given text string, causing the agent to speak. *SMBShakeNod* causes the agent to shake or nod her head, with a specified count, period, and amplitude.

Finally, *SMBToyAction* is an *SMBBase* subclass that interfaces with some code written especially for a toy-sorting task used in child development studies. In this task, toys are presented in pairs, with a large and a small version of the same toy in each pair. This is triggered from the state machine using the “Present Pair” action on the *SMBToyAction* component. On either side of the table is a bucket, one large and one small. One of the toys is indicated by moving it closer to the user, who is then asked which way it should go (done via the “Ask To Sort” action). When the user indicates left or right (detected via “user:intent:isPushLeft” and “user:intent:isPushRight” keys on the blackboard), then the toy moves into the corresponding bucket via the “Sort Left” and “Sort Right” actions.

Figure 4.2 shows a basic implementation of this protocol. The lowermost seven states all make use of the *SMBToyAction* component to trigger manipulations of toys on the table. Some of them

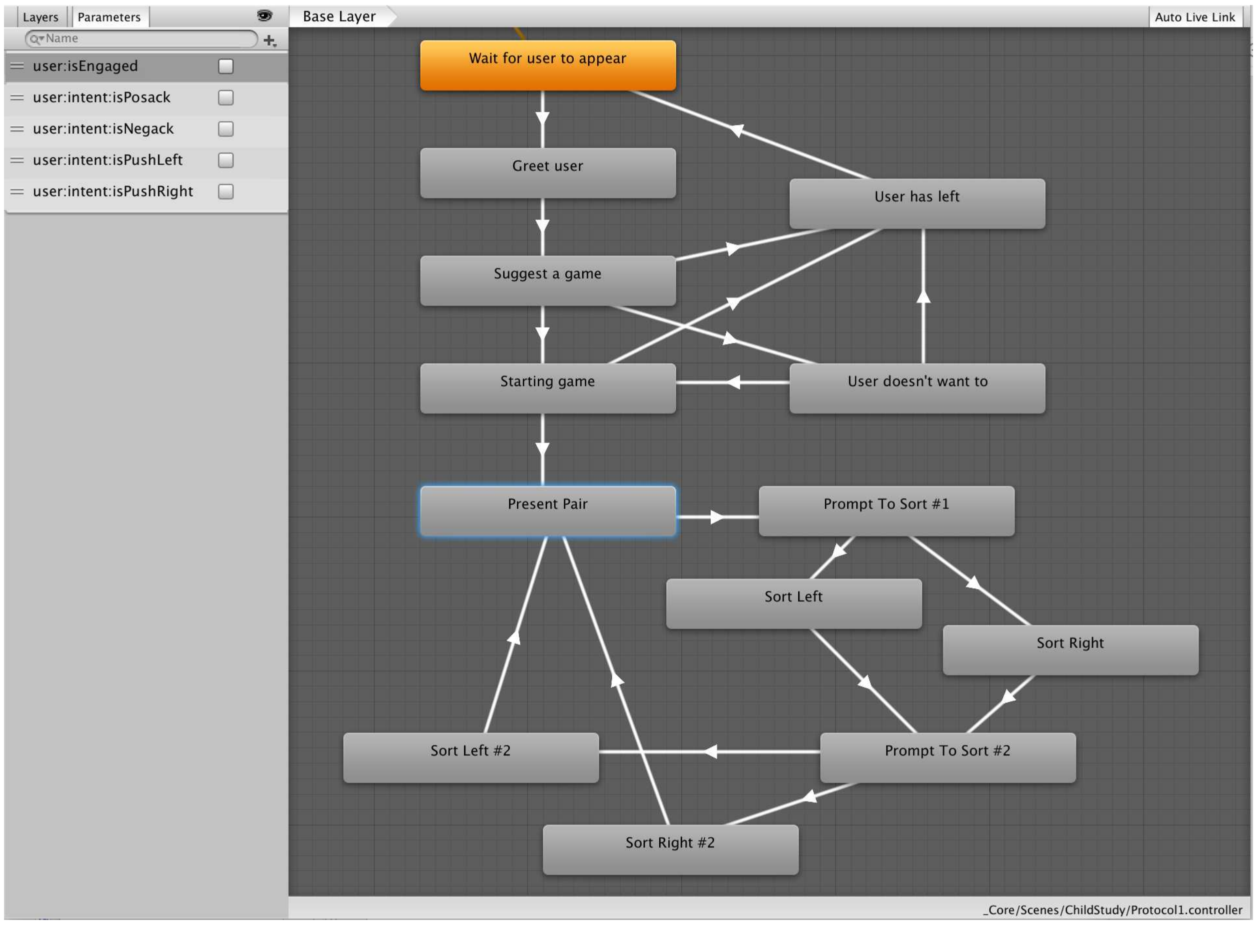


Figure 4.2: A state machine implementing the toy-sorting protocol. Blackboard keys used are shown as state-machine parameters on the left.

also make use of *SMBSpeak* and *SMBShakeNod* to provide verbal and nonverbal feedback to the user.

4.2 RGB gesture perception

One of the design goals for the Faelyn system was to be more easily portable, i.e., to work without a dedicated setup and specialized sensor hardware. The input needs for developmental science studies are minimal; we only need the children to indicate either the left or right bucket. Accordingly, we have developed a gesture detection module based on optical flow that can detect gross left and right motions.

This module, implemented as the C# script *OpenCVFlow*, is built on OpenCV, or more specifically on a C# port called OpenCvSharp. This calculates dense optical flow across the image using the Farneback algorithm [50], and then finds the mean horizontal movement across the whole image, then averaged across a window of frames. When the absolute value of this averaged movement exceeds a threshold, it is signaled as a "left" or "right" motion. The detector then goes into a refractory (cooldown) period during which it will not signal either direction again; this helps prevent rebound signals which otherwise often occur at the end of a gesture. Reasonable values for these parameters were found empirically to be: window = 5 frames, threshold = 0.6; cooldown = 20 frames.

The *OpenCVFlow* script is written independent of the rest of the Diana/Faelyn system. To interface it with the cognitive architecture, an *OpenCVPushModule* was added, which sets the "user:intent:isPushLeft" and "user:intent:isPushRight" keys on the blackboard according to the current signal from the detector.

Together, these scripts enable the Faelyn system to respond to push-left and push-right gestures on any computer equipped with an ordinary RGB camera.

4.3 Initial user test

Finally, an early version of the Faelyn Fox system was used for an informal gesture elicitation study in late January, 2020 at the Early Childhood Education Center in Fort Collins. In this study, children of 5-7 years age (with caregiver consent) were brought into the library one at a time to interact with the system running on a laptop computer. A "wizard of Oz" technique was employed whereby this author stood some distance behind the child, operating the system via an unobtrusive wireless gamepad. In this version of the system, one toy was presented at a time, though still with two buckets of different sizes as shown in Figure 1.2. Another experimenter (Dr. Daunhauser) explained that small toys should go in the small bucket, and big toys should go in the big bucket. The children were given instructions such as "Faelyn can see your hands. Use your hands to show which bucket the toy should go in." They were carefully *not* told exactly how to do so.

Each toy was then presented, with the experimenter giving the child a prompt such as "here's a big sheep. Where should that go?" After sorting toys in this manner for several minutes, the children were told, "Now let's play a silly game. Let's put the big toys in the small bucket, and the small toys in the big bucket!" Sorting then continued under this new rule. Data were recorded in several ways: events were logged to a file by the Unity code; written observations were taken by Dr. Ortega; and both RGB and depth-sensor video was captured by the Kinect camera.

The purpose of this study was to see what gestures children would use to indicate each bucket, when not prompted to do this in any particular way. The result was a very large and varied gesture set. The rest of this section presents some key observations.

The first child immediately reached for the screen, and started using it like an iPad: with one finger extended, she touched the toy on the screen, and tried to drag it to the correct bin. When the toy did not track her finger fast enough, she did it again, as if the software had simply failed to track properly. One might speculate that this child had prior experience with touch interfaces. For subsequent trials, the screen was moved back from the edge of the table, the child's chair was moved away, and the children were instructed to stay seated.

One child pointed very clearly, with a whole arm extension. It was noticed that the arm and hand would be extended first, somewhat tentatively, and then when a decision was made, the index finger was extended and the arm did a jabbing motion, making it very clear that she had decided.

Another child did a variety of swiping gestures, with varying speeds, orientation of the hands, etc. Even within the broad category of "swipe" (a horizontal motion of the arm/hand in the desired direction), the highly variable gestures observed would likely present a challenge to an automated classifier.

One child used a reverse-bloom "grasping" gesture on the toy, then moved her hand towards one bucket or the other, and did a "release" gesture. She also tried virtually grasping the toy, and then throwing it left or right (and seemed pleased when this worked).

One child kept his hands in his lap, and did very subtle side-to-side gestures with the index finger, or sometimes with the whole hand. These were difficult even for the human operator to confidently perceive.

Another child used pointing, but instead of pointing at the bucket, pointed first at the toy, and then traced an arc similar to the desired path of the toy on screen (having picked up on the toy-to-bucket animation shape after only a few trials). She was essentially interacting as if with a touch interface, trying to "drag" the toy where she wanted it, even though pointing at a screen several feet away.

When we did the "silly game," reversing the rule about where the toys go, it occasionally happened that a child made a mistake and put a toy in the wrong bucket. Though no feedback was given, I think in every case the child immediately noticed their error, and looked quickly to the experimenter as if to see what they should do about it. In such cases the experimenter would say something like "keep going!" and the child would resume sorting correctly. It was also observed that response times were longer in the silly game.

These preliminary results suggest several insights for future work. First, unconstrained gestures from children this age are extremely variable. To make an automated detector and classifier for such gestures would be a technical challenge. A better approach might be to prime the children to make one particular type of gesture, such as pointing or swiping, and have them practice a few times in a simpler task, such as getting Faelyn to look left or right. A more complete elicitation study could determine which sort of gesture would feel natural to most children. Second, the “silly game” did indeed appear to be an appropriate challenge to children of this age, requiring some extra thought (measured as slower response time). Careful measurement of this response time could serve as a useful metric, providing insight into developmental stage and executive function, specifically cognitive flexibility (personal communication, Lisa Daunhauer).

Chapter 5

Conclusions

5.1 Directions for future work

There are ample opportunities for future research related to the Faelyn Fox system; some of these relate to the technical qualities of the system, while others are opportunities for new tasks that might be useful in assessing child development.

First, a proper gesture elicitation study could be performed to gather data on what sort of gestures children use to indicate the bucket to which a toy should go. Video recordings could be manually classified, and the corresponding Kinect skeleton pose data could be used to program or train detector algorithms for the most common or useful gestures. Note that in our early user test (see section 4.3), children found the animated arc traced by the toys to be attention-grabbing, and often shaped their gestures to imitate that motion. A future gesture elicitation study might want to instead have Faelyn pick up and move the toy in a more natural manner, perhaps with some random variation in the path taken.

The Kinect sensor provides reliable pose estimation, and with the help of existing code and models in the CwC lab, is a demonstrated way to detect a wide variety of user gestures, including pointing. But it requires substantial physical setup that may be inconvenient for some contexts in which Faelyn Fox could be useful. So, pose estimation and gesture perception based on an ordinary RGB camera may be worth pursuing. The current Faelyn Fox system has a very simple way to detect gross left/right gestures based on optical flow, but this code could probably be fine-tuned and further improved. To take it further, real-time RGB pose estimation is needed. Several libraries in recent years are able to do this task at reasonable framerates on some hardware, e.g. MoVNect [51].

At some point it may be valuable for Faelyn (or Diana) to be able to perceive not only the pose of the user, but also the position and orientation of physical objects on the table, providing a truly symmetric capability, i.e. one where both Faelyn and the human user can perceive objects on both sides of the table, though only manipulate their own. This might be possible using a library such as MobilePose [52].

Along similar lines, eye/gaze tracking could be of value, particularly in working with younger children where such data can be used to measure attention and delayed recognition. 3D eye gaze tracking using a single RGB camera has been demonstrated [53].

With such perceptual abilities, Faelyn Fox would be capable of other tasks besides the current toy-sorting task. For example, there is a bear/dragon task in which children are supposed to follow instructions from the bear but ignore instructions from the dragon; this tests inhibitory self-control [54]. There are also tasks that test children's understanding of false belief; for example, Faelyn could place a toy in a box, leave the scene or turn away, and then let the children move the toy to the other box. Faelyn would then turn back, and the children would be asked where Faelyn thinks the toy is, and where the toy actually is. Such tasks test the development of a theory of mind, which is affected in DS, autism, and Asperger syndrome [55] [56].

Finally, as noted in section 3.6, correct parsing of natural language input in the context of a cooperative agent is greatly hampered by the fact that current text corpuses are mainly culled from news articles, blog posts, and tweets, all of which consist of almost entirely declarative statements. Declarative statements have a different grammatical structure than the interrogative and imperative statements most frequently used when working with an agent. Development of a corpus of interrogative and imperative sentences, tagged with part of speech and/or parse data, would be a valuable step towards building better parsers for actually communicating with computers, and a very helpful service to the community.

5.2 Discussion

Early AI projects, such as the Hearsay speech understanding model in the 1970s [43], faced several issues. To solve complex problems, many knowledge sources were needed, each applying different heuristics and transformations to the data. The work was necessarily experimental in nature, prompting researchers to enable some knowledge sources, disable others, and modify still others; to do this effectively required those modules to remain loosely coupled. There was also a desire to decompose the system to take advantage of multiple processors. At the same time, each module needed to expand on and correct the hypotheses made by other modules.

These apparently conflicting requirements — modules completely independent yet able to work together on a problem — were solved via the blackboard architecture. A central blackboard served as the data bus connecting modules which had no direct communication or interdependencies. This solution proved quite successful, and led to a series of advances in understanding speech, and became a cornerstone of AI in the late 20th century; the main architectural paper [57] has been cited over 2000 times.

In the early days of AI there was some tension between symbolic (logic-based) and connectionist (artificial neural network) approaches, but as Marvin Minsky pointed out in 1991 [58], successful AI systems are often built out of diverse components, some connectionist and some symbolic. Yet in recent years the astounding success of "deep learning" (large neural networks trained on very large datasets) has taken the world by storm, to the extent that one may be tempted to see them as the “master algorithm” that can do everything. Yet contemporary researcher Pedros Domingos, in his book *Master Algorithm* [59], identifies these as only two of five “tribes” of AI (the others are genetic algorithms, Bayesian logic, and analogical modeling), and argues that none of these can solve all the problems of AI on its own. Like Minsky, Domingos believes that a human-like AI system is likely to need techniques from all these approaches.

The Diana agent faced issues of complexity, interdependency, and fragility very similar to those tackled by the designers of Hearsay in the 1970s. In the course of this Diana 2.0 project, we found that a very similar solution still applies today. We expanded on it; unlike the Hearsay blackboard, which shared only hypothesis data, ours is used for the complete pipeline of perceptual, cognitive, and intention data. But the core idea is the same. At the same time, unlike Hearsay but as predicted by Minsky, our Diana agent also makes use of deep learning (the connectionist approach) on perceptual tasks for which such systems are especially well suited.

While projects like Diana press forward on the artificial intelligence side, other developments in technology are foreshadowing a future in which embodied conversational agents are commonplace. After several false starts over the decades, virtual reality (VR) appears to finally be here to stay, with consumer devices such as Oculus Quest and HTC Vive becoming commonplace. Augmented reality (AR) is at an earlier stage, with HoloLens and Magic Leap selling mainly to developers or enterprised customers for specialized purposes. However each generation of these is better than the last, and presistent rumors swirl that Apple may be announcing AR glasses that may do for AR what the iPod did for MP3 players, or the iPhone did for cell phones. We will live in a world where our computers can manifest realistic images and sounds in both virtual worlds and in the real one.

In both VR and AR, you generally have no easy access to a keyboard; any communication with other agents (whether human or artificial) is primarily through speech and gesture. Consumer non-embodied agents such as Alexa and Google Home manage to serve useful functions with speech only, out of technological necessity; but once VR and AR become sufficiently common, such agents will surely present themselves in an embodied form, and take advantage of the richer multimodal communication such form affords. The roles an intelligent ECA could perform in the future are innumerable: personal assistant, medical advisor, search agent, or research assistant, just to name a few. These roles will surely include working with children, allowing human experts to more quickly and consistently assess developmental progress, and even provide interventions such as social training.

These ECAs of the future will almost certainly combine both symbolic and connectionist approaches, just as Diana does today. In this project, I've played a part in bringing a key idea from the early days of AI into a modern context, and made it a core component of a revolutionary computer interface. With Diana 2.0 and Faelyn Fox on a solid architectural footing, they push forward toward the day when intelligent ECAs are our steadfast companions.

Bibliography

[1] Nikhil Krishnaswamy, Pradyumna Narayana, Isaac Wang, Kyeongmin Rim, Rahul Bangar, Dhruva Patil, Gururaj Mulay, Ross Beveridge, Jaime Ruiz, Bruce Draper, et al. Communicating and acting: Understanding gesture in simulation semantics. In *IWCS 2017 — 12th International Conference on Computational Semantics — Short papers*, 2017.

[2] Robbert-Jan Beun, Eveliene De Vos, and Cilia Witteman. Embodied conversational agents: effects on memory performance and anthropomorphisation. In *International Workshop on Intelligent Virtual Agents*, pages 315–319. Springer, 2003.

[3] Preben Wik and Anna Hjalmarsson. Embodied conversational agents in computer assisted language learning. *Speech communication*, 51(10):1024–1037, 2009.

[4] Silvia Tamayo-Moreno and Diana Pérez-Marín. Designing and evaluating pedagogic conversational agents to teach children. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering*, 11(3):488–493, 2017.

[5] Rohit Kumar and Carolyn P Rose. Architecture for building conversational agents that support collaborative learning. *IEEE Transactions on Learning Technologies*, 4(1):21–34, 2010.

[6] Alexis Bosseler and Dominic W Massaro. Development and evaluation of a computer-animated tutor for vocabulary and language learning in children with autism. *Journal of autism and developmental disorders*, 33(6):653–672, 2003.

[7] Hiroki Tanaka, Hideki Negoro, Hidemi Iwasaka, and Satoshi Nakamura. Embodied conversational agents for multimodal automated social skills training in people with autism spectrum disorders. *PloS one*, 12(8), 2017.

[8] Beatriz López Mencía, David Díaz Pardo, Alvaro Hernández Trapote, and Luis A Hernández Gómez. Embodied conversational agents in interactive applications for children with special educational needs. 2014.

[9] Lisa A Daunhauer, Brianne Gerlach-McDonald, Elizabeth Will, and Deborah J Fidler. Performance and ratings based measures of executive function in school-aged children with down syndrome. *Developmental neuropsychology*, 42(6):351–368, 2017.

[10] Elizabeth A Will, Lisa A Daunhauer, Deborah J Fidler, Nancy Raitano Lee, Cordelia Robinson Rosenberg, and Susan L Hepburn. Sensory processing and maladaptive behavior: Profiles within the down syndrome phenotype. *Physical & occupational therapy in pediatrics*, 39(5):461–476, 2019.

[11] Lisa A Daunhauer, Elizabeth Will, Emily Schworer, and Deborah J Fidler. Young students with down syndrome: Early longitudinal academic achievement and neuropsychological predictors. *Journal of Intellectual & Developmental Disability*, pages 1–11, 2020.

[12] Jane Clifford O'Brien, Harriet G Williams, Anita Bundy, Jim Lyons, and Amita Mittal. Mechanisms that underlie coordination in children with developmental coordination disorder. *Journal of Motor Behavior*, 40(1):43–61, 2008.

[13] Monique Natalie Beutum, Reinie Cordier, and Anita Bundy. Comparing activity patterns, biological, and family factors in children with and without developmental coordination disorder. *Physical & Occupational Therapy in Pediatrics*, 33(2):174–185, 2013.

[14] Bruno Dumas, Denis Lalanne, and Sharon Oviatt. *Multimodal Interfaces: A Survey of Principles, Models and Frameworks*, pages 3–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[15] Susan R Fussell, Leslie D Setlock, Jie Yang, Jiazhi Ou, Elizabeth Mauer, and Adam DI Kramer. Gestures over video streams to support remote collaboration on physical tasks. *Human-Computer Interaction*, 19(3):273–309, 2004.

[16] Alex Lascarides and Matthew Stone. Formal semantics for iconic gesture. In *Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue (BRANDIAL)*, pages 64–71, 2006.

[17] Donald A. Norman. Natural user interfaces are not natural. *Interactions*, 17(3):6–10, May 2010.

[18] Casey Kennington, Spyridon Kousidis, and David Schlangen. Interpreting situated dialogue utterances: an update model that uses speech, gaze, and gesture information. *Proceedings of SigDial 2013*, 2013.

[19] Joakim Gustafson, Nikolaj Lindberg, and Magnus Lundeberg. The august spoken dialogue system. In *Sixth European Conference on Speech Communication and Technology*, 1999.

[20] Justine Cassell, Yukiko I Nakano, Timothy W Bickmore, Candace L Sidner, and Charles Rich. Non-verbal cues for discourse structure. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 114–123. Association for Computational Linguistics, 2001.

[21] Catherine Pelachaud, Valeria Carofiglio, Berardina De Carolis, Fiorella de Rosis, and Isabella Poggi. Embodied contextual agent in information delivering application. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 758–765, 2002.

[22] Wolfgang Wahlster. Towards symmetric multimodality: Fusion and fission of speech, gesture, and facial expression. In *Annual Conference on Artificial Intelligence*, pages 1–18. Springer, 2003.

[23] Stefan Kopp, Bernhard Jung, Nadine Lessmann, and Ipke Wachsmuth. Max-a multimodal assistant in virtual reality construction. *KI*, 17(4):11, 2003.

[24] Arthur C Graesser, Haiying Li, and Carol Forsyth. Learning by communicating in natural language with conversational agents. *Current Directions in Psychological Science*, 23(5):374–380, 2014.

[25] Stefan Kopp and Ipke Wachsmuth. Synthesizing multimodal utterances for conversational agents. *Computer animation and virtual worlds*, 15(1):39–52, 2004.

[26] Stefan Kopp, Lars Gesellensetter, Nicole C Krämer, and Ipke Wachsmuth. A conversational agent as museum guide–design and evaluation of a real-world application. In *International workshop on intelligent virtual agents*, pages 329–343. Springer, 2005.

[27] Elisabeth André and Catherine Pelachaud. Interacting with embodied conversational agents. In *Speech technology*, pages 123–149. Springer, 2010.

[28] Isaac Wang, Pradyumna Narayana, Dhruva Patil, Gururaj Mulay, Rahul Bangar, Bruce Draper, Ross Beveridge, and Jaime Ruiz. Exploring the use of gesture in collaborative tasks. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2990–2997, 2017.

[29] Pradyumna Narayana, Nikhil Krishnaswamy, Isaac Wang, Rahul Bangar, Dhruva Patil, Gururaj Mulay, Kyeongmin Rim, Ross Beveridge, Jaime Ruiz, James Pustejovsky, and Bruce Draper. Cooperating with avatars through gesture, language and action. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Systems and Applications*, pages 272–293, Cham, 2019. Springer International Publishing.

[30] Max M Louwerse, Arthur C Graesser, Danielle S McNamara, and Shulan Lu. Embodied conversational agents as conversational partners. *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, 23(9):1244–1255, 2009.

[31] Timothy W Bickmore, Laura M Pfeifer, Donna Byron, Shaula Forsythe, Lori E Henault, Brian W Jack, Rebecca Silliman, and Michael K Paasche-Orlow. Usability of conversational agents by patients with inadequate health literacy: evidence from two clinical trials. *Journal of health communication*, 15(S2):197–210, 2010.

[32] Cynthia Matuszek, Liefeng Bo, Luke Zettlemoyer, and Dieter Fox. Learning from unscripted deictic gesture and language for human-robot interactions. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[33] Sean Andrist, Michael Gleicher, and Bilge Mutlu. Looking coordinated: Bidirectional gaze mechanisms for collaborative interaction with virtual characters. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 2571–2582, 2017.

[34] Bilge Mutlu, Jodi Forlizzi, and Jessica Hodgins. A storytelling robot: Modeling and evaluation of human-like gaze behavior. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 518–523. IEEE, 2006.

[35] Ewa Luger and Abigail Sellen. " like having a really bad pa" the gulf between user expectation and experience of conversational agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5286–5297, 2016.

[36] Mikyong Shin, Lilah M Besser, James E Kucik, Chengxing Lu, Csaba Siffel, Adolfo Correa, et al. Prevalence of down syndrome among children and adolescents in 10 regions of the united states. *Pediatrics*, 124(6):1565–1571, 2009.

[37] Fiorenza Stagni, Andrea Giacomini, Sandra Guidi, Elisabetta Ciani, and Renata Bartesaghi. Timing of therapies for down syndrome: the sooner, the better. *Frontiers in behavioral neuroscience*, 9:265, 2015.

[38] Susan R Harris, Elizabeth CR Mickelson, and Jill G Zwicker. Diagnosis and management of developmental coordination disorder. *Cmaj*, 187(9):659–665, 2015.

[39] Stephanie M Carlson, Louis J Moses, and Laura J Claxton. Individual differences in executive functioning and theory of mind: An investigation of inhibitory control and planning ability. *Journal of experimental child psychology*, 87(4):299–319, 2004.

[40] Nancy Bayley and Glen Aylward. The bayley scales of infant development (4th ed.). 2019.

[41] M. Rhonda Folio and Rebecca R. Fewell. Peabody developmental motor scales, 2nd edition. 2000.

[42] Susan A Rose, Judith F Feldman, and Jeffery J Jankowski. Implications of infant cognition for executive functions at age 11. *Psychological science*, 23(11):1345–1355, 2012.

[43] Lee D. Erman. Overview of the hearsay speech understanding research. *SIGART Bull.*, (56):9–16, February 1976.

[44] H Penny Nii. The blackboard model of problem solving and the evolution of blackboard architectures. *AI magazine*, 7(2):38–38, 1986.

[45] Stuart Maclean. On the singleton software design pattern. Technical report, Technical Report DSSE-TR-97-4, Dept of Electronics and Computer Science . . . , 1997.

[46] Redis. https://redis.io, 2020. [Online; accessed 25-May-2020].

[47] Heting Wang. An empathic avatar in task-driven human-computer interaction. Master's thesis, Colorado State University, Fort Collins, Colorado, 2020.

[48] W Nelson Francis and Henry Kucera. Brown corpus manual. *Letters to the Editor*, 5(2):7, 1979.

[49] Rada Mihalcea. SemCor 3.0. http://www.nltk.org/nltk_data/, 2020. [Online; accessed 25-May-2020].

[50] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[51] Dong-Hyun Hwang, Suntae Kim, Nicolas Monet, Hideki Koike, and Soonmin Bae. Lightweight 3d human pose estimation network training using teacher-student learning. *arXiv preprint arXiv:2001.05097*, 2020.

[52] Tingbo Hou, Adel Ahmadyan, Liangkai Zhang, Jianing Wei, and Matthias Grundmann. Mobilepose: Real-time pose estimation for unseen objects with weak shape supervision. *arXiv preprint arXiv:2003.03522*, 2020.

[53] Congyi Wang, Fuhao Shi, Shihong Xia, and Jinxiang Chai. Realtime 3d eye gaze animation using a single rgb camera. *ACM Transactions on Graphics (TOG)*, 35(4):1–14, 2016.

[54] Marjorie A Reed, Diana L Pien, and Mary K Rothbart. Inhibitory self-control in preschool children. *Merrill-Palmer Quarterly (1982-)*, pages 131–147, 1984.

[55] Nurit Yirmiya, Daphna Solomonica-Levi, Cory Shulman, and Tammy Pilowsky. Theory of mind abilities in individuals with autism, down syndrome, and mental retardation of unknown etiology: The role of age and intelligence. *Journal of Child Psychology and Psychiatry*, 37(8):1003–1014, 1996.

[56] Atsushi Senju, Victoria Southgate, Sarah White, and Uta Frith. Mindblind eyes: an absence of spontaneous theory of mind in asperger syndrome. *Science*, 325(5942):883–885, 2009.

[57] Lee D Erman, Frederick Hayes-Roth, Victor R Lesser, and D Raj Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys (CSUR)*, 12(2):213–253, 1980.

[58] Marvin L. Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 12(2):34, Jun. 1991.

[59] Pedro Domingos. *The Master Algorithm: How the quest for the ultimate learning machine will remake our world.* Basic Books, Lebanon, IN, 2015.