THESIS

PRACTICAL ASPECTS OF DESIGNING AND DEVELOPING A MULTIMODAL EMBODIED AGENT

Submitted by

Rahul Bangar

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2021

Master's Committee:

Advisor: Ross Beveridge
Co-Advisor: Francisco R. Ortega

Christopher Peterson

ABSTRACT

PRACTICAL ASPECTS OF DESIGNING AND DEVELOPING A MULTIMODAL EMBODIED AGENT

This thesis reviews key elements that went into the design and construction of the CSU CwC Embodied agent, also known as the Diana System. The Diana System has been developed over five years by a joint team of researchers at three institutions – Colorado State University, Brandeis University and the University of Florida. Over that time, I contributed to this overall effort and in this thesis, I present a practical review of key elements involved in designing and constructing the system. Particular attention is paid to Diana's multimodal capabilities that engage asynchronously and concurrently to support realistic interactions with the user.

Diana can communicate in visual as well as auditory modalities. She can understand a variety of hand gestures for object manipulation, deixis, etc and can gesture in return. Diana can also hold a conversation with the user in spoken and/or written English. Gestures and speech are often at play simultaneously, supplementing and complementing each other. Diana conveys her attention through several non-verbal cues like slower blinking when inattentive, keeping her gaze on the subject of her attention, etc. Finally, her ability to express emotions with facial expressions adds another crucial human element to any user interaction with the system.

Central to Diana's capabilities is a blackboard architecture coordinating a hierarchy of modular components, each controlling a part of Diana's perceptual, cognitive, and motor abilities. The modular design facilitates contributions from multiple disciplines, namely VoxSim/VoxML with Text-to-speech/Automatic Speech Recognition systems for natural language understanding, deep neural networks for gesture recognition, 3D computer animation systems, etc. – all integrated within the Unity game engine to create an embodied, intelligent agent that is Diana. The primary contribution of this thesis is to provide a detailed explanation of Diana's internal working along with a thorough background of the research that supports these technologies.

ACKNOWLEDGEMENTS

I am deeply grateful to my advisors, Dr. Ross Beveridge and Dr. Francisco R. Ortega, for guiding me throughout the completion of this work. Their words of encouragement have been a constant source of motivation for me, especially in my times of doubt. I'd also like to thank Dr. Bruce A. Draper for giving me the opportunity to learn from the best at Computer Vision lab. Ever since I joined, I've always found him available whenever I've needed help. And when things would get hectic and tense, he would always have a funny remark to keep everyone in good spirits. I'm also thankful to all my wonderful colleagues from whom I learned a lot. I'd like to especially thank Dr. Pradyumna Narayana for always introducing me to interesting problems that we'd have a blast racking our brains over, for helping me whenever I got stuck, and for always pushing me to be better. It is also my humble honor to have worked with some of the nicest people like Jason Yu, Dhruva Patil, Gururaj Mulay, David White, Heting Wang, and Matt Dragan and I couldn't thank them all enough for all the support.

I'm hugely indebted to my parents for always believing in me and encouraging me to follow my dreams. Thanks Dad, for always being there to guide me. And Mom, for the unconditional love and for being patient whenever I've found myself frustrated. And *Nani*, though you're no longer with us, I'll always cherish the memories of you playing the rubber duck to my long technical ramblings that would almost always improve my own understanding.

Last but not the least, I deeply appreciate the help and cordiality provided by the Department of Computer Science at CSU that made it a remarkable place to learn and work at.

DEDICATION

I would like to dedicate this thesis to my mentors, my parents, and my friends.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Human face-to-face communication is always multimodal. This means that such communication involves multiple sensory and multiple production modalities, where each modality corresponds to one of the five human senses of sight, hearing, touch, smell, and taste [2]. Since people react socially to computers [3], there has been a trend to anthropomorphize the human-computer interface. This has been further reinforced with the advent of ubiquitous computing, where our surroundings have become smarter, and traditional computer interfaces are no longer sufficient. In their stead, Embodied Conversational Agents (ECAs), also called Virtual Humans, have emerged as a viable multimodal social interface that simulate human face-to-face communication. ECAs are computer models resembling humans in their bodily look and their communication capabilities. They are able to perceive and understand both the virtual environment they live in and the real world in which the human resides.

Diana is an embodied conversational agent developed by the people in Computer Vision Lab at Colorado State University in collaboration with the brilliant minds at Lab for Linguistics and Computation at Brandeis University and Ruiz Human-Computer Interaction Lab at University of Florida. Funded by DARPA's Communicating With Computers program, the project's vast scope meant it received invaluable contributions from a significant number of people. James Pustejovsky and Nikhil Krishnaswamy's work on multimodal simulations from language expressions drives Diana's natural language understanding and reasoning via the interpretive machinery provided by VoxML [4] and VoxSim [5]. Pradyumna Narayana contributed the ResNet-style deep convolutional neural networks [6] that allow Diana to predict gestures accurately in real time which are core to Diana's multimodal perception capabilities. However, the aforementioned Deep Neural Network's accuracy is also owed to the massive multimodal dataset, EGGNOG [7]. EGGNOG was collected during human-to-human elicitation studies conducted to better understand communicative gestures used by humans in the context of a collaborative

task. Labeling the dataset with gesture labels was again a joint effort carried out by Jamie Ruiz, Issac Wang, Gururaj Mulay, Dhruva Patil, and the author of this thesis. The task of labeling was facilitated by EASEL [8] — a tool that eases the annoation of gesture video datasets by providing automatic segmentation [9] for gestures. This reduces the burden of the labeling task. A major improvement to Diana's usability is due to her ability to map user's pointing gesture to a location in her virtual environment, whose implementation is credited to Jason Yu. Diana also benefits from having a blackboard architecture at her core which is implemented by Joe Strout that ensures responsiveness and asynchrony in Diana's behaviors. Heting Wang's work [10] on adding empathic facial expressions to Diana add another layer to Diana's personality. The author's contributions to the system have ranged across a variety of areas, which include writing the networking code for inter-component communication in the system as well as to serve all the streams captured by Microsoft Kinect in real time. Additionally, the author implemented the state machine architecture that combines the noisy signals from each individual perceptual component, and translates them into a stable one that can be used by Diana. Over time, the system has amassed significant codebase and its maintenance responsibilities have been shared over the years by the author, Nikhil Krishnaswamy, David White, Dhruva Patil, and Matt Dragan.

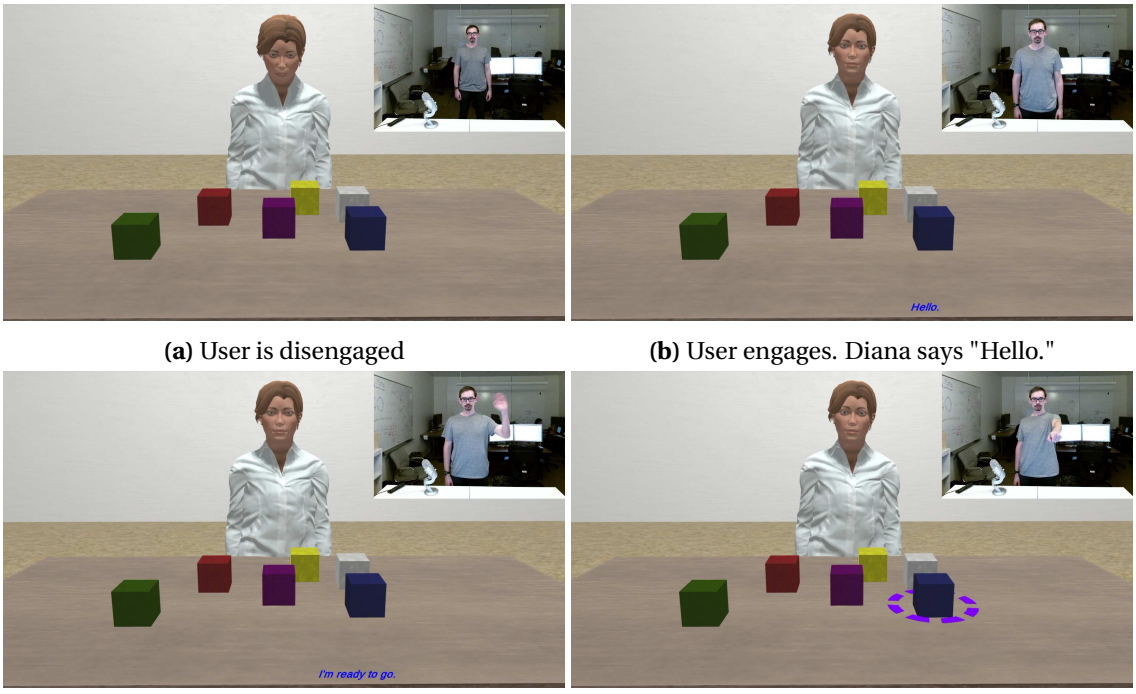
Diana has a multitude of capabilities that allow her to act intelligently. She can see the user, and realize when the user is standing up close and is engaged as opposed to when the user is standing far away and disengaged. Her visual senses are not limited to just depth perception, however. She can also understand hand gestures. Her visual perception of the world transitions seamlessly from the real to the virtual. In the virtual world, she can see objects like blocks, the table, etc. She understands the bounds of the virtual table, and can intelligently figure out a path for moving the blocks around.

Diana can also talk to the user and engage in a meaningful discourse. She has the ability to recognize and understand complete sentences. The user can specify an action verbally, and Diana will be able to carry it out. Diana also realizes when the user asks a question, and can

Figure 1.1: Typical steps to coordinate a staircase construction with Diana

reply appropriately based on her knowledge of the world. The discourse is not limited to turn taking either. Much more realistically, Diana is able to respond to interruptions immediately without needing to wait for the current action to be completed. Recognizing that emotions are an essential human quality, Diana can recognize when the user smiles, when user displays concentration on the task at hand, or when the user gets frustrated. She can also express emotions via facial expressions when appropriate.

All these capabilities together serve to provide Diana with an extremely anthropomorphic human-machine interface.



(a) User is disengaged

(b) User engages. Diana says "Hello."

(c) User waves. Diana says "I'm ready to go."

(d) User points at the blue block

Figure 1.2: Building a staircase. A detailed deconstruction of the working of agent internals for this task is presented in the next chapter.

1.1 Building a Staircase

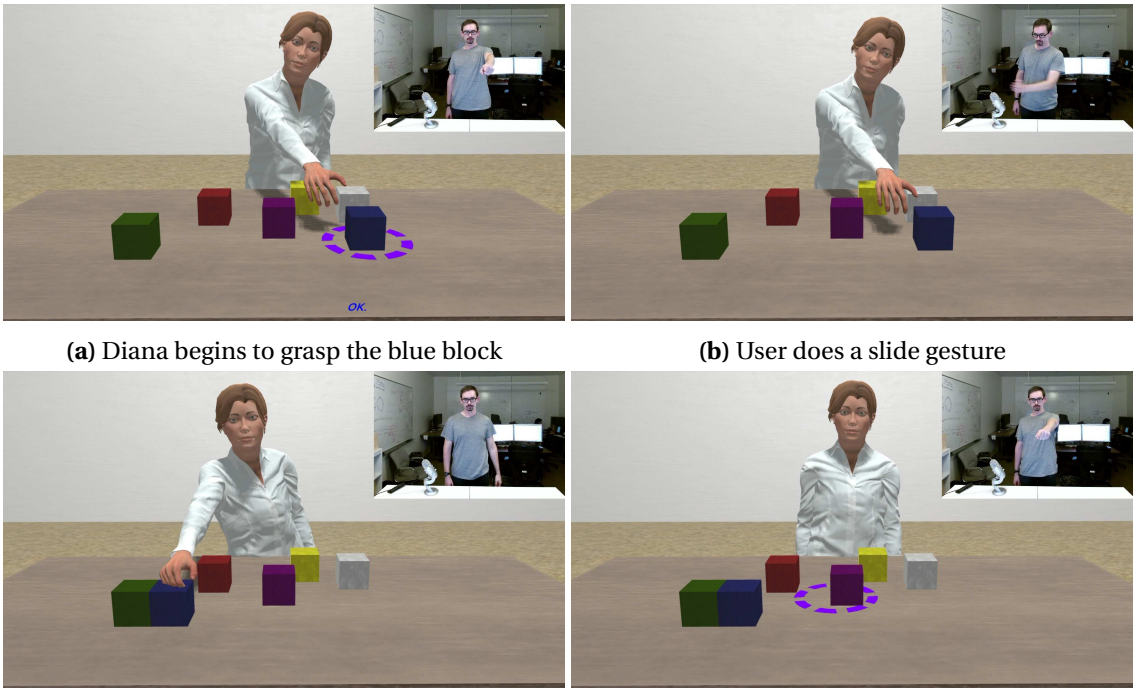
In this section, we will discuss Diana's capabilities in greater detail with a walkthrough of a common scenario of "staircase building". Although simplistic from a human viewpoint, the scenario allows us to display the variety of Diana's capabilities and their interplay with each other. The scene consists of Diana, a virtual table on which six virtual blocks are placed. Each virtual block is of a distinct color, namely red, blue, green, yellow, purple, and white. This scenario involves the following steps:

1. User walks up to the system (Figure 1.2a-1.2b).

2. Diana realizes that the user is standing near to the system, and greets by saying "Hello." (Figure 1.2b).

3. The user waves with either hand (Figure 1.2c).

4. Diana waves back, and replies "I'm ready to go." (Figure 1.2c).



(a) Diana begins to grasp the blue block

(b) User does a slide gesture

(c) Diana slides the blue block

(d) User points at the purple block

Figure 1.3: Building a staircase

5. User points at the screen. A visual marker appears on the virtual table identifying the location being pointed at (Figure 1.2d).

6. User adjusts the pointing until the visual marker lies on one of the virtual blocks, the blue one (Figure 1.2d).

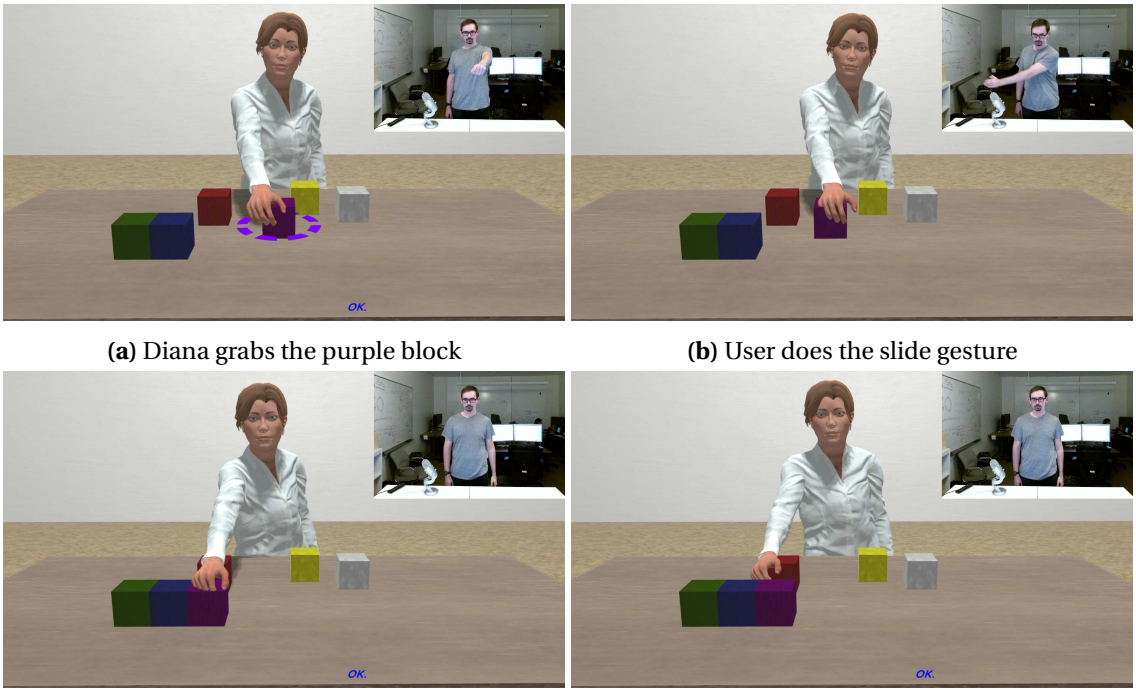
7. Holding the visual marker stationary over the blue block for a short time, Diana recognizes that the user means to select it. Realizing that, Diana says "OK." as she begins to grasp the blue block (Figure 1.3a).

8. The user does a slide gesture with either hand (Figure 1.3b).

9. Diana slides the blue block against the green block, and retreats back into the resting idle pose (Figure 1.3c).

10. User points at the purple block (Figure 1.3d).

11. Diana says "OK." as she begins to grasp the purple block (Figure 1.4a).



(a) Diana grabs the purple block

(b) User does the slide gesture

(c) Diana slides the purple block

(d) User says "the red one." Diana grabs the block.

Figure 1.4: Building a staircase

12. The user does a slide gesture with either hand (Figure 1.4b).

13. Diana slides the purple block against the previously slid blue block, and retreats back into the resting idle pose (Figure 1.4c).

14. User says "the red one" (Figure 1.4d).

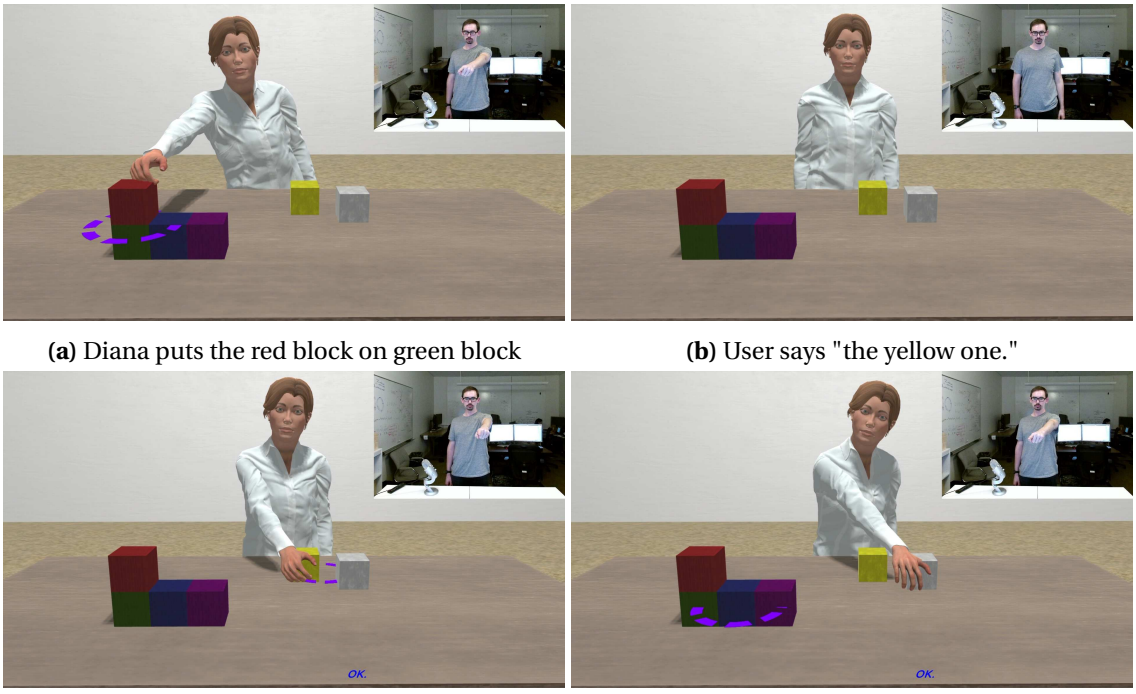
15. Diana understands the phrase to mean the red block, says "OK." to acknowledge her understanding, and begins to grasp the red block (Figure 1.4d).

16. User points at the green block (Figure 1.5a).

17. Diana puts the red block on top of the green block (Figure 1.5a).

18. User tries to point to the white block, but accidentally holds the visual marker over the nearby placed yellow block (Figure 1.5b).

19. Diana says "OK." and begins to grasp the yellow block (Figure 1.5b).



(a) Diana puts the red block on green block

(b) User says "the yellow one."

(c) User interrupts "No, the white one."

(d) Diana says "OK." and grabs white block instead

Figure 1.5: Building a staircase

20. User says "No, the white block" (Figure 1.5c).

21. Diana realizes the interruption and instead of continuing the motion to grasp the yellow block, she adjusts it to grasp the white block instead, while acknowledging her understanding of interruption by saying "OK" (Figure 1.5d).

22. User points at the blue block (Figure 1.5d).

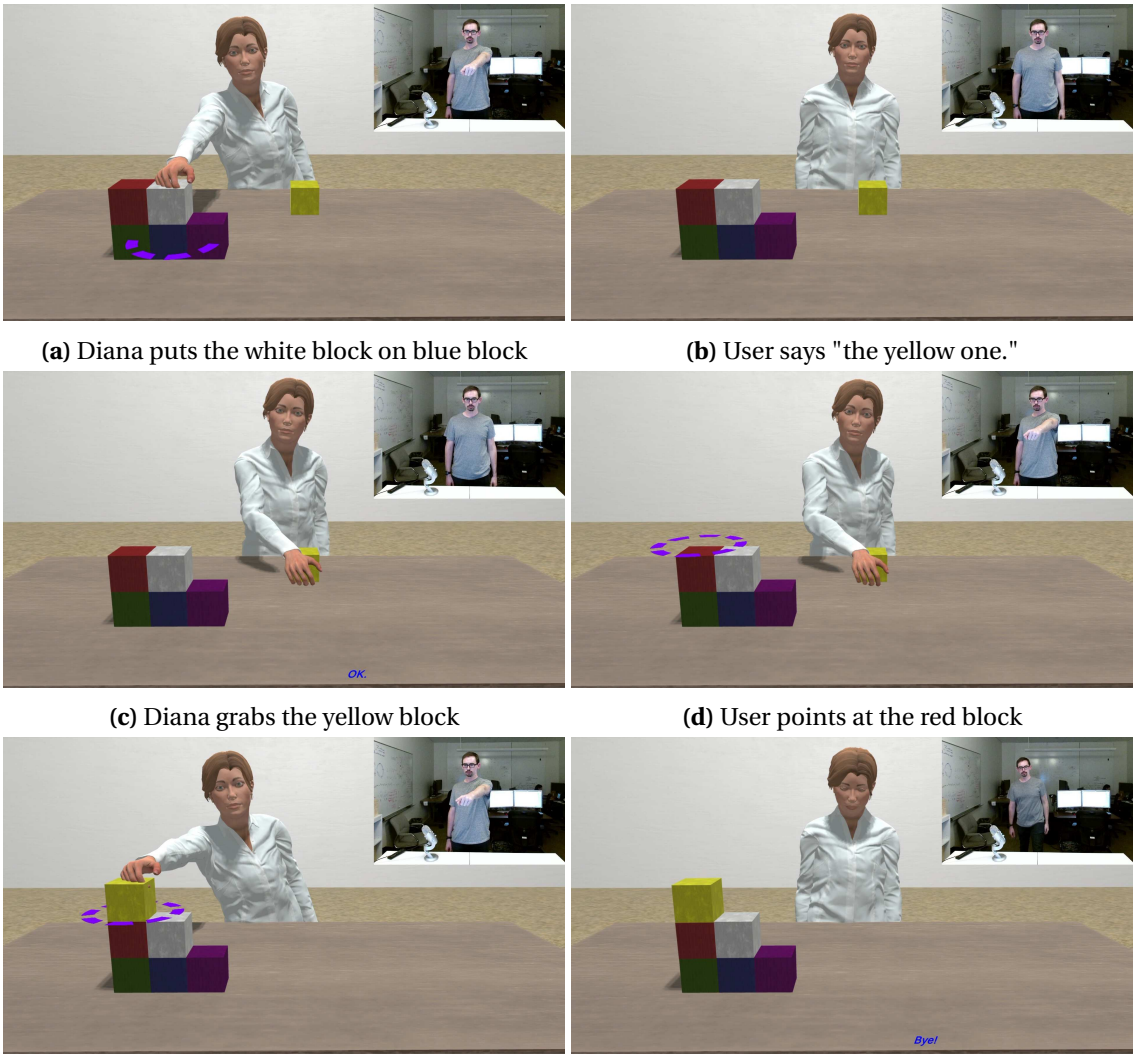
23. Diana puts the white block on top of the blue block (Figure 1.6a).

24. User says "the yellow one" (Figure 1.6b).

25. Diana understands the phrase to mean the yellow block, says "OK." to acknowledge her understanding, and begins to grasp the yellow block (Figure 1.6c).

26. User points at the red block (Figure 1.6d).

27. Diana puts the yellow block on top of the red block (Figure 1.6e).



(a) Diana puts the white block on blue block

(b) User says "the yellow one."

(c) Diana grabs the yellow block

(d) User points at the red block

(e) Diana puts the yellow block on red block

(f) User steps back. Diana says "Bye."

Figure 1.6: Building a staircase

28. User says "Bye." and retreats away from the system (Figure 1.6f).

29. Diana realizes that the user has stepped away and says "Bye" (Figure 1.6f).

1.2 Abilities

The previous section showcases many of Diana's abilities. The following is a complete list of all the abilities that Diana has along with a brief description.

Gaze Allows directing Diana's gaze to any arbitrary point or a specific object (virtual or real).

Blinking Controls Diana's implicit need to blink that is proportional to changes in attention level.

Grasp Allows Diana to grasp an object or point to an arbitrary location.

Attention Controls Diana's object of attention that may change in response to user actions.

Alertness Controls Diana's alertness level depending on her ongoing activity and whether she is focusing on something or not.

Speech Allows Diana to synthesize speech.

Speech Recognition Allows Diana to transcribe user speech into text.

Language Understanding Allows Diana to parse and understand text into object references and actions on them.

Emotion Controls Diana's facial expressions.

Gesture Recognition Ability to recognize hand gestures.

Arm Motion Recognition Ability to recognize arm motions and the direction of the motion.

Affect Ability to recognize user's emotions like smiling, frustration, etc.

Engagement Diana recognizes when the user walks up to the system as being "engaged" as opposed to stepping back or staying away from the system as "disengaged"

Pointing Diana can understand when the user is pointing at the screen and extrapolate the pointing gesture from real world coordinate system into the virtual world's coordinate system where it is displayed by a visual marker

Sliding Intent Diana can recognize when the user does a swiping motion with either hand to mean horizontal movement.

Servo Intent Diana can recognize when the user does a servo motion with either hand to mean slight adjustments.

Positive Acknowledgement Intent Diana can recognize when the user does a thumbs up gesture with either hand to mean confirmation of a query.

Negative Acknowledgement Intent Diana can recognize when the user does a thumbs down gesture with either hand to mean refutation of a query.

Grabbing Intent Diana can recognize when the user does a claw gesture with either hand to mean grabbing an object.

Push Intent Diana can recognize when the user does a swiping motion towards the display to mean pushing an object towards her.

Pull Intent Diana can recognize when the user does a swiping motion towards himself/herself to mean pushing an object towards the user (i.e. pulling the object away from Diana).

Counting Intent Diana can recognize when the user signals a count using fingers.

Stop Intent Diana can recognize when the user raises the hand in a stop gesture to mean stop an ongoing action immediately.

Slide Action Diana can slide a block sideways across the table, usually against another block.

Carry Action Diana can carry a block from one location to another, usually on top of another block.

Note that in the above list, we have differentiated between gestures and intents. At the surface, both seem similar. However, intents are a higher level abstractions that are composed from one or more gestures and arm motions over time. For example, the sliding intent is composed from the hand being in a specific posture and the arm swiping across the table sideways.

1.3 Motivation

The usefulness of creating a system such as Diana lies in the abilities that she possesses. As a multimodal avatar, Diana represents a state-of-the-art integration that merges sight, speech and a shared perception of both the user's body and a shared world which is partially virtual. The abilities themselves are implemented in a modular way so that each ability can be selectively enabled or disabled, can be finely controlled with parameters, or even replaced with an overriding implementation.

Gaze provides an important non-verbal cue in mediating human-human communication. One study [11] found that avatars with informed gaze significantly outperform those with random gaze across multiple response variables. Diana also uses her gaze for different non-verbal cues. She can express her attention through gaze. When the user is not close to the system, Diana will avert her gaze to peripheral locations in the virtual environment. When the user is engaged with the system, Diana will begin to look at the user instead. When interacting with objects, Diana follows the object of her attention with her gaze.

Diana is situated [12] [13] in her environment. This means that her understanding of meaning is closely tied to the environment she resides in. If a user gives a command that is impossible to complete due to the conditions imposed by the environment, Diana will be able to reason why the command can't be completed and suggest alternatives to the user. For example, if block B is on top of block A and the user asks Diana to put block C on top of block A, Diana can infer that block A's top is occupied and hence it cannot "afford" to have any other block on top unless block B is removed first. Therefore, she can suggest so to the user and the user can proceed appropriately.

Being multimodal, Diana already surpasses current generation's voice assistants with regards to the communication possibilities. For example, saying something like "What is that?" to Google Home, the voice assistant replies with "Sorry, what are you asking?". Simply because the agent in this case neither has the modality to find an answer to this question, nor is the agent

situated in its environment. Diana, on the other hand, checks both boxes. Therefore, Diana can be used in more challenging environments and roles than many other voice assistants.

For example, one can imagine Diana teaching a user how to disassemble and reassemble an appliance. The transition from manipulating blocks to manipulating screws, coverings, etc. (virtual, of course) isn't that huge of a leap. On the plus side, though, Diana need not work on a physical machine to teach someone, whereas the current analogous method would be to have a video of someone operating on a physical machine. Moreover, a video is static in the sense that if it could have a camera orientation or point-of-view that may obstruct the component being discussed from user's view. Diana, on the other hand, lives in a simulation and the point of view in the virtual world can be readily adapted to match user's needs, thereby offering a more dynamic tutorial than a video.

Chapter 2

Related Work

Diana has a multitude of capabilities that combine various research disciplines into a concrete usable system. The purpose of this chapter is to briefly discuss the literature on each of these research domains. Starting with Section 2.1, we describe the work on dialogue systems. Diana's architecture is backed by a blackboard system to ensure modularity, which is discussed in Section 2.2. The natural language understanding and reasoning component of the system is grounded in visualization supported by VoxSim/VoxML platform as discussed in Section 2.3. The system also employs gesture recognition using deep learning networks trained on the gesture data gathered in EGGNOG dataset, both of which are described in Section 2.4. Diana is an embodied agent, a trait that has been shown to be useful in a variety of domains as mentioned in Section 2.5. One of the important abilities that Diana has is to understand deictic gestures through her ability to pinpoint a location pointed at by the user. Section 2.6 briefly describes the different pointing techniques used by various systems.

2.1 Dialogue Systems

Dialogue is a conversation between two or more participants. While the participants are usually human, intelligent systems can be designed to serve as a stand-in for a human participant in a dialogue. Such systems are called Dialogue systems.

Traditionally, dialogue systems have been limited to typed input and output owing to the technical limitations of the day, even though spoken language has been recognized as a more time-efficient input modality as compared to traditional input modalities like keyboard and mouse [14]. Although the serial nature of spoken language output can make the response generated by a dialogue system feel dragged out, it does lend a naturalness to the conversation that typed or even displayed output simply cannot. With recent advances in speech recognition and text-to-speech synthesis, dialogue systems are now easily equipped with the ability to listen to

user's commands and communicate the results back by speaking in turn. Such dialogue systems are called Spoken dialogue systems. These systems have become increasingly popular in recent times through the use of digital assistants like Siri, Alexa, Google Assistant, etc. that people employ to manage their daily tasks.

Speech, however, is just one of the modalities that humans use to communicate. Modalities like gestures, facial expressions, body postures, gaze, etc. offer information that can supplement speech very well. For example, a spoken sentence like "It's there." isn't very meaningful unless one also points at a location. In this case, the referential nature of gestures aids spoken language to convey a useful intent. Utilizing this interdependence of modalities is crucial to reducing ambiguity in the dialogue and increasing user accessibility of the system. The dialogue systems that utilize more than one modality for both input and output are called Multimodal dialogue systems.

Based on the techniques used in their implementation and their intended application, dialogue systems are classified [15] into two types:

1. Task-oriented Systems

2. Non-task-oriented Systems

Note that this classification is not mutually exclusive and some systems have characteristics of both types. Non-task oriented systems are simpler, so we discuss them first followed by an overview of task-oriented systems.

2.1.1 Non-task-oriented Systems

Non-task-oriented systems assist the user in unstructured, open domain conversations. These are also called chatbots. The three common architectures that are used to implement chatbots are discussed in the following sections, concluding with a brief description of ELIZA [16], a well-known chatbot system.

Rules and patterns

The first architecture, and probably the simplest one, uses patterns and templates connected by transform rules. The patterns are matched against the input. The pattern contains zero or more placeholders which assume a value corresponding to the placeholder's position in the pattern. The output is created by employing the transform rule for the matched pattern to select the correct template and the missing fields in the template are filled in by the values assumed by the corresponding placeholders. As an example, when the transform rule

He [$placeholder_1$] me → Why do you think he [$value_1$] you?

is matched against the input "He loves you.", the system recognizes that the pattern matches the input and $placeholder_1$ holds the value *loves*, so the transform rule is applied while replacing $value_1$ with *loves*. The generated output is, therefore, "Why do you think he *loves* you?"

Information Retrieval

Information retrieval-based systems rely on a huge dataset of human-to-human dialogue of input-output pairs. Briefly, the system reads the user input, matches it against the available data, and generates the output. This can be done via two different strategies.

The first strategy matches the user input against all the inputs in the dataset and selects from amongst them the input that matches the most with the user input. It then proceeds to choose the response corresponding to matched input in the dataset. In short, the strategy is to return the response to the most similar input.

The second strategy is very similar to the first one, except that it directly returns the input that matches the most with the user input instead of choosing the corresponding response from the dataset. Choosing the most similar input from the dataset means that it is likely to be lexically and semantically similar to user input, and hence, a good candidate for a reasonable response. Although a bit less intuitive than the previous strategy, this, in fact, seems to work better in practice.

Deep Learning

Instead of devising an enormous number of rules and patterns, deep learning models can be utilized, provided a huge amount of training data is available. Then the model learns all the rules automatically and in a much more comprehensive and sophisticated manner than would be possible with hand-engineering.

While deep neural networks have been successful in a variety of domains, the sequential and temporal nature of language is well-suited to recurrent neural networks (RNNs) [17], a generalization of feed-forward neural networks to sequential input. The input is represented by a sequence of real valued vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$. RNNs model the sequential input by maintaining a hidden state at each step – a memory of what the network has seen so far. Given the weight matrices W_{xh}, W_{hh}, and W_{hy}, the hidden state $\mathbf{h}_t$ and output $\mathbf{y}_t$ at each step (or time) t are determined as:

$$\mathbf{h}_t = \sigma(W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{y}_t = \text{softmax}(W_{hy}\mathbf{h}_t + \mathbf{b}_y)$$

Here, $\mathbf{b}_h$ and $\mathbf{b}_y$ are bias terms.

Unfortunately, RNNs suffer from the problem of vanishing and exploding gradients, and in general, struggle to learn long range dependencies [18] [19]. To overcome these issues, Long Short Term Memory (LSTM) [20] were designed. In LSTMs, the nodes of RNNs are replaced by memory cells. A memory cell is constructed in a very specific way that consists of an input node, gates and multiplicative nodes which mitigates RNNs' limitations as mentioned above.

In a memory cell, the input node $\mathbf{g}_t$, the input gate $\mathbf{i}_t$, the forget gate $\mathbf{f}_t$ and the output gate $\mathbf{o}_t$ in the LSTM $\mathbf{g}_t$ act on the current input $\mathbf{x}_t$ along with hidden state from the previous step $\mathbf{h}_{t-1}$ and wrap the weighted sum inside a non-linearity like tanh or *sigmoid* function. The internal state is a sum of the outputs of input node and the internal state at previous timestep, both gated by input gate and forget gate respectively. Gating is accomplished by pointwise multiplication

(multiplicative nodes) of a gate's output with a node's output. The input gate learns when to let the input activation into the internal state. Similarly, the output gate $\mathbf{o}_t$ learns when to let the value out of the cell. Finally, the forget gate learns when to erase the effects of internal state at the previous timestep $\mathbf{s}_{t-1}$. The memory cell is described by the following equations [21]:

$$\mathbf{g}_t = \sigma(W_{gx}\mathbf{x}_t + W_{gh}\mathbf{h}_{t-1} + \mathbf{b}_g)$$

$$\mathbf{i}_t = \sigma(W_{ix}\mathbf{x}_t + W_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = \sigma(W_{fx}\mathbf{x}_t + W_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{o}_t = \sigma(W_{ox}\mathbf{x}_t + W_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

$$\mathbf{s}_t = \mathbf{g}_t \odot \mathbf{i}_t + \mathbf{s}_{t-1} \odot \mathbf{f}_t$$

$$\mathbf{h}_t = \tanh(\mathbf{s}_t) \odot \mathbf{o}_t$$

Since LSTMs can tackle long-range dependencies very well, they are suitable for sequence to sequence mapping. Seq2Seq [22] architecture utilizes two recurrent neural networks, LSTMs specifically, to accomplish this. Given the input sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, <\text{EOS}>)$, one of the LSTMs – the encoder – acts on each input vector, updating its hidden state $\mathbf{h}_t$ at each step. The value of hidden state at the last step is stored as the context vector $\mathbf{c}$ for the second LSTM – the decoder. The decoder LSTM takes this hidden state $\mathbf{c}$ as an input while managing a hidden state $\mathbf{h'}_t$ of its own. The last output of encoder $\mathbf{y}_T$ is fed into the decoder as the first input $\mathbf{x'}_1$. Thereafter, each output of decoder $\mathbf{y'}_t$ is fed back to the memory cell for the next time step as the input $\mathbf{x'}_{t+1}$. In this way, the encoder-decoder architecture predicts the sequence $(\mathbf{y}_T, \mathbf{y'}_1, \ldots, \mathbf{y''}_T, <\text{EOS}>)$.

An Example: ELIZA

ELIZA [16] was an early natural language processing system created at MIT Artificial Intelligence Laboratory. The user interacts with the system by typing a sentence in, and the system in turn types the response out.

The system behavior is driven by scripts. The scripts describe keywords and their associated transformation rules. The program can accept scripts written by the user, so the behavior is customizable.

Given a script, the system functions by looking for a keyword in the input sentence. The input may consist of delimiters like comma (,) and period (.), in which case it is split along these delimiters into individual parts and the parts are scanned from left to right for a keyword. The search stops at the first part in which a keyword is found. This is to ensure that only single sentences are transformed.

A transformation rule is composed from a decomposition rule and a reassembly rule. To illustrate the formal notation used to write these rules, an example follows:

(0 YOU 1 ME) → WHY DO YOU THINK I 3 YOU?

The left hand side is the decomposition rule. 0 represents any number of words. A positive integer n represents exactly n words. The right hand side is the reassembly rule. In a reassembly rule, the integers represent the index of the constituent of the decomposition rule. To illustrate the use of this transformation rule, suppose the user said "I think you hate me." Then the system matches 0 to "I think" and 1 to the single word "hate". It then applies the transformation rule to generate "Why do you think I hate you?" where 3 is the index of the placeholder 1 in decomposition rule.

Keywords have a precedence so that less general words have a higher precedence than the more general ones. In this way, the transformation rules associated with higher ranked keywords are triggered more often, enabling the system to generate more interesting responses.

A typical conversation with ELIZA as recorded in [16] is given below. The capitalized sentences are generated by ELIZA.

Men are all alike.

IN WHAT WAY

They're always bugging us about something or other.

CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.

YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time

I AM SORRY TO HEAR YOU ARE DEPRESSED

It's true. I am unhappy.

DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY?

I need some help, that much seems certain.

WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP

Perhaps I could learn to get along with my mother.

TELL ME MORE ABOUT YOUR FAMILY

My mother takes care of me.

WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

My father.

YOUR FATHER

You are like my father in some ways.

WHAT RESEMBLANCE DO YOU SEE

You are not very aggressive but I think you don't want me to notice that.

WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE

You don't argue with me.

WHY DO YOU THINK I DON'T ARGUE WITH YOU

You are afraid of me.

DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU

My father is afraid of everybody.

WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER

Bullies.

DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE

By today's standards, ELIZA's design is incredibly simple, yet its pattern-action architecture is sophisticated enough that according to Weizenbaum, people sometimes ended up being deeply emotionally involved with the program. For example, there's a story that one of Weizenbaum's staff would request to be allowed to talk to ELIZA privately. Although Weizenbaum suggested that he might need to store all the conversation data for later analysis, people were hesitant because of the privacy implications, and would point it out as such. It didn't matter to people that ELIZA was just a program, as they considered it "real" enough to have private conversations with it. It is, perhaps, no surprise therefore that ELIZA's pattern-action is still used by modern chatbots.

2.1.2 Task-oriented Systems

Task-oriented systems, unlike chatbots, are focused on helping the user accomplish a specific task. As such, they tend to have a much deeper understanding of a specific domain at the expense of a narrower domain. Historically, task-oriented systems have also been distinguished from chatbots by their use of natural language processing techniques as compared to much simpler rules/patterns architecture used in chatbots, but that distinction doesn't hold true anymore.

Since a task-oriented dialogue tends to be highly structured, understanding the structure underlying such a conversation has been a primary focus for researchers. [23] argues that many of the utterances not only convey information, but also perform an action. For example, an utterance like "John, can you close the door?" isn't really a question about John's ability to close the door. Instead, it is a directive asking John (politely) to close the door instead. Understanding

language in terms of these acts through language – Speech Acts – allows one to take a transactional view of dialogue that can be much simpler to model than trying to exhaustively model the world of the participants [24].

Some of the assumptions about Speech Acts may not be true, however, when viewed in a broader scope of dialogue as noted in [25]. Particularly, the effect of the listener being unable to understand the speaker's utterance is not considered. [25] propose that the "act" of a Speech Act only takes place after the listener has understood it, and hence, the single-agent speech acts are replaced by multi-agent, collaborative, grounded actions called Dialogue Acts [26]. Dialogue acts depend on the task domain at hand.

In addition to modeling dialogue structure, the issue of knowledge representation in such systems is very important. Frames [27] are a data structure that can represent chunks of information. A frame consists of slots. The slots, in turn, can have semantically typed values like City, Date, etc. Some values are simple, like Integer. More complicated value types can be represented as frames themselves. Prototype-based inheritance allows one to abstract the more complicated frames in a hierarchy. The prototype is a template to create an object, and each instance of a prototype has a reference to its prototype object, thereby establishing a hierarchy chain. This hierarchy chain is resolved at runtime to link a particular slot to the correct parent frame's slot. The frames as proposed by [27] also allowed two types of procedures to be attached to frames. The first type of procedure would be run whenever a slot was changed. This is analogous to *setter*s of modern languages like C#. The second type of procedures were meant to be called explicitly and are analogous to *methods* in modern object-oriented languages.

Two predominant architectures are used to implement task-oriented systems [28]. We discuss these next, followed by a description of SHRDLU – a well-known task-oriented system.

Pipeline based

In a pipeline based system, the system is divided into distinct components connected in a serial fashion, with the output of one component serving as the input to the next component, hence the name. The components are named as follows:

- **Natural Language Understanding** This component handles many responsibilities that are focused on the understanding of the user utterance. For example, domain and intent classification of an utterance are performed by this component. Domain classification categorizes the user utterance to a specific domain, while the intent classification further categorizes an utterance within a given domain to a specific intention of the user. For example, an utterance like "What flights are available from Denver to New York?", the system should be able to recognize the domain as "AIR TRAVEL" and the intent as "RESERVATION". Given these two broader classifications, the system can apply an appropriate model to fill the slots (slot-filling) of the corresponding "FLIGHT" frame, say "FROM-CITY", "TIME", etc.

  CMU's Phoenix Natural Language Understanding system [29] uses a set of grammars for each slot such that the rules map word patterns to slot type. More recently, deep learning techniques like Recurrent Neural Networks (RNNs) [17] [30] have also been used for slot-filling. Similar data-driven techniques have also been employed for domain and intent classification.

- **Dialogue State Tracker** This component is responsible for predicting the user's goal given the dialogue state H_t at time t where the state is represented as a frame. Hand-crafted rules are one of the simpler ways to accomplish this. More sophisticated approaches maintain a probability distribution over the various values allowed by a slot's type instead.

- **Dialogue Policy Learning** Taking as input the current user goal as tracked by the Dialogue State Tracker, this component is responsible for generating an appropriate system action. Supervised as well as reinforcement learning can be used to optimize this component.

- **Natural Language Generation** Given the action generated by the Policy Learning component, this component is responsible for converting this internal representation into a fluent, easy to understand response in human language. Recurrent Neural Networks like LSTMs have been utilized to perform response generation.

End-to-end based

Since the pipeline-based systems have components that are dependent on each other, every time a component is replaced, the other components have to be re-optimized as well to fit in the new pipeline. Additionally, it is hard to relate user feedback to a specific component in the pipeline. As a result, more recent advances have been made by training a neural network directly on the corpus of annotated dialogue datasets, with the problem being formulated as that of training a network to learn a mapping of dialogue state history to system response. An end-to-end system like this has no dependence on additional components, and can be replaced in its entirety if needed.

An example: SHRDLU

SHRDLU [31] was a natural language processing system developed by Terry Winograd during his doctoral research at M.I.T. in 1968-70. One of the major intuitions guiding the design of SHRDLU is that to understand language, one should understand the syntax and semantics along with planning in an integrated way. It proposed that an utterance can be understood to be triggering appropriate procedures in the listener's cognitive system for each constituent of the utterance, and all these procedures – whether for understanding the syntax or the semantics or for planning – would all run concurrently sharing information amongst themselves freely.

The domain in which SHRDLU operated is called Blocks World. Blocks World is one of the oldest application domain in artificial intelligence in which researchers addressed human-computer interaction and planning. Briefly, it consists of a number of blocks of possibly different shapes, sizes, and colors placed on a table, and the goal is to turn an initial block configuration into a goal configuration, by moving one block at a time. Blocks can be stacked on top of each other. Blocks world serves as a useful surrogate for cooperative tasks where the partners share a workspace. In case of SHRDLU, the user types the commands or queries. The systems displays the simulated scene of the blocks world on a display, and types out responses or requests for information.

Knowledge is represented in SHRDLU in Micro-Planner [32] language. Elementary facts about the world are stored as assertions. For example, "(IS X BLOCK)" represents the fact that the object referred to by the symbolic name "X" is a "BLOCK" – the symbolic name of the concept of the block. More complex concepts, for example concepts that may change the state of the world are stored as longer Micro-Planner procedures. For example, the concept of clearing the top of a block i.e. "CLEARTOP" could be represented as:

(GOAL (IS X? BLOCK))

(GOAL (SUPPORT X? Y?))

(GOAL (GET-RID-OF Y?))

As can be seen from the above example, a given procedure can call other procedures, or even itself recursively. The variables are denoted by the ? suffix. The program substitutes the variables by the objects in the world, and notes that a choice has been made when a substitution takes place. The choice is important to note, because the program can backtrack to the point where the choice is made in case the current choice doesn't lead to a successful proof of goal. When backtracking to a previous point, the program makes a new choice and keeps on attempting to prove the goal. The goal is proved when all the individual goals in the list are proved. The stacking of choices is important, since it allows the program to answer "How" and "Why" questions by examining the stack history. The fact that Micro-Planner has primitives that make looping and backtracking implicit makes it an intuitive choice for representing concepts.

The semantic definitions of concepts are also written as procedures in a similar way. For example, the following procedure defines a cube as an equidimensional, manipulable block:

```
(CUBE
    ((NOUN (OBJECT
        (MANIPULABLE RECTANGULAR) ((IS ? BLOCK) (EQUIDIMENSIONAL ?))
    )))
)
```

The syntax is analyzed in terms of Systemic Grammar [33]. Systemic Grammar describes a hierarchy of *syntactic units*. A syntactic unit at a given level of hierarchy is composed of syntactic units below its level. Each syntactic unit has features describing it, as well as the functions it may fill in syntactic units of which it is a constituent. These features form a structured system of choices, hence the name systemic grammar. In SHRDLU, procedures written in PROGRAMMAR language operate on an input string to classify words into syntactic units and a valid input will only allow specific combination of features.

A sample conversation between SHRDLU and the user provided by [34] is given below. The capitalized sentences are generated by SHRDLU.

> Pick up a big red block.
>
> OK.
>
> Grasp the pyramid.
>
> I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
>
> Find a block which is taller than the one you are holding and put it into the box.
>
> BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.
>
> OK.
>
> What does the box contain?
>
> THE BLUE PYRAMID AND THE BLUE BLOCK.
>
> What is the pyramid supported by?
>
> THE BOX.
>
> How many blocks are not in the box?
>
> FOUR OF THEM.
>
> Is at least one of them narrower than the one which i told you to pick up?
>
> YES, THE RED CUBE.
>
> Is it supported?
>
> YES, BY THE TABLE.

Can the table pick up blocks?

NO.

Can a pyramid be supported by a block?

YES.

Can a pyramid support a pyramid?

I DON'T KNOW.

Stack up two pyramids.

I CAN'T.

Will you please stack up both of the red blocks and either a green cube or a pyramid?

OK.

Which cube is sitting on the table?

THE LARGE GREEN ONE WHICH SUPPORTS THE RED PYRAMID.

Is there a large block behind a pyramid?

YES, THREE OF THEM: A LARGE RED ONE, A LARGE GREEN CUBE, AND THE BLUE ONE.

Put a small one onto the green cube which supports a pyramid.

OK.

Put the littlest pyramid on top of it.

OK.

Does the shortest thing the tallest pyramid's support supports support anything green?

YES, THE GREEN PYRAMID.

What color is it?

BY "IT", I ASSUME YOU MEAN THE SHORTEST THING THE TALLEST PYRAMID'S SUPPORT SUPPORTS.

RED

How many things are on top of green cubes?

I'M NOT SURE WHAT YOU MEAN BY "ON TOP OF "IN THE PHRASE "ON TOP OF GREEN CUBES " .

DO YOU MEAN:

1 - DIRECTLY ON THE SURFACE

2 - ANYWHERE ON TOP OF ?

2

THREE OF THEM.

Had you touched any pyramid before you put the green one on the little cube?

YES, THE GREEN ONE.

When did you pick it up?

WHILE I WAS STACKING UP THE RED CUBE, A LARGE RED BLOCK AND A LARGE GREEN CUBE.

Why?

TO GET RID OF IT.

Why did you do that?

TO CLEAN OFF THE RED CUBE.

Why did you clear off that cube?

TO PUT IT ON A LARGE GREEN CUBE.

Why did you do that?

TO STACK UP THE RED CUBE , A LARGE RED BLOCK AND A LARGE GREEN CUBE.

Why did you do that?

BECAUSE YOU ASKED ME TO.

How did you do it?

BY PUTTING A LARGE RED BLOCK ON THE TABLE ; THEN LETTING GO OF IT ; THEN PUTTING A LARGE GREEN CUBE ON IT ; THEN LETTING GO OF THAT CUBE ; THEN PUTTING THE RED CUBE ON THAT CUBE; THEN LETTING GO OF THAT CUBE.

How many objects did you touch while you were doing it?

FOUR OF THEM.

What did the red cube support before you started to clean it off?

THE GREEN PYRAMID.

There were five blocks to the left of the box then.

NO, ONLY FOUR OF THEM: THE RED CUBE, TWO LARGE GREEN CUBES AND A LARGE RED BLOCK.

Put the blue pyramid on the block in the box.

OK.

Is there anything which is bigger than every pyramid but is not as wide as the thing that supports it?

YES, THE BLUE BLOCK.

Does a steeple

Computer (interrupting): SORRY, I DON'T KNOW THE WORD "STEEPLE".

A "steeple" is a stack which contains two green cubes and a pyramid.

I UNDERSTAND.

Are there any steeples now?

NO.

Build one.

OK.

Call the biggest block "superblock".

OK.

Have you picked superblock up since we began?

YES.

Why did you drop it?

BY "IT", I ASSUME YOU MEAN THE SUPERBLOCK.

I CAN'T EXPLAIN A NON-EXISTENT EVENT.

Is there anything to the right of the red pyramid?

YES, FOUR THINGS: THE BOX , THE RED CUBE, THE BLUE PYRAMID AND THE BLOCK NAMED SUPERBLOCK

Thank you.

YOU'RE WELCOME!

SHRDLU is related to Diana in multiple ways, the most basic of which is that both operate in the prototypical environment of blocks world. The idea of linguistic utterances triggering specific cognitive processes modeled as programs is not entirely carried over. However, there is a resemblance to motion verbs being parsed into a predicate-argument set which facilitates a procedurally generated visual simulation of an event. Both Diana and SHRDLU are situated in their environment, meaning they are aware of their surroundings, specifically the blocks, and take it into account while inferring meaning. For example, both will refuse an action, if a particular block configuration is not feasible. Similarly, both will suggest alternatives in case of an ambiguity. For Diana, VoxML [4] adds rich semantic information to the objects, including the concept of scale and color, which allow her to differentiate objects based on these criteria, much like SHRDLU.

That said, there are major differences too, though it may not be apparent at the surface. Diana's interpretive machinery is based on VoxML, which is rooted in Generative Lexicon [35]. SHRDLU's semantic understanding is based on a grammar that decomposes meaning, while Generative Lexicon distributes semantic load on all the constituents of an utterance. Moreover, SHRDLU's perception capabilities are not sensor based and entirely programmed, whereas Diana can actually perceive the world in a visual as well as aural modality in addition to the text-only interface that SHRDLU has.

2.2 Blackboard Systems

Imagine a group of experts sitting in a room with a blackboard and mulling over a complex problem. They are not allowed to talk to each other. Instead, each expert is given a chalk to record their "contribution" to the solution on the blackboard. A contribution is essentially part of the solution that reflects expert's inference according to his area of expertise. Other experts in the room can use these "contributions", provided it falls within their area of expertise, to arrive at their own "contributions". Some of these contributions naturally build on each other to form progressively higher level contributions to the solution. In this scenario, the experts might as well not be aware of any other expert and how they came about their contribution; all they know is the blackboard. The independent nature of experts means that if one of them is unavailable for some reason, say illness, the group can still arrive at a solution albeit of a lower quality.

This metaphor, credited to [36] [37], describes the essence of blackboard architecture systems. First used in HEARSAY-II [38] for speech understanding, the architecture has been used to solve a diverse range of problems like sonar interpretation [39], errand planning [40], protein structure analysis [41], etc. in the years since.

In terms of implementation, the blackboard takes the form of a globally accessible database consisting of entries. An entry is a set of key-value pairs and represents the "contribution" in the metaphor described previously. Entries in the database are arranged in a linear hierarchy to reflect the various abstraction levels to attack the problem. The set of keys belonging to an entry constitute its vocabulary. The vocabulary of an entry depends on the problem being solved as well as the level of the entry. It is also possible to have a uniform vocabulary i.e. all entries have the same set of keys. For example, HEARSAY-II [38]. Entries can be linked to each other to form a more cohesive solution. For example, a higher level entry linked to a lower level entry may mean that that the higher level entry is justified by the information in the lower one.

The experts in the metaphor are represented by Knowledge Sources, which are simply event handlers that can subscribe to change of an existing entry, addition of a new entry, or satisfaction of a logical predicate involving entries. When triggered, the event handlers may add,

modify or delete entries from the blackboard. Like any event driven system, a scheduler lies at its core which determines what event handlers to trigger given the current blackboard state. Event handlers may be executed concurrently. Moreover, an existing event handler may be interrupted to favor another event handler if it is expected to lead to a better solution. Such a system is said to exhibit opportunism.

2.3 Visually Grounded Reasoning

People engaging in a conversation are bound to have certain beliefs and presuppositions even before the conversation begins. Some of these are assumed to be common knowledge, while others are supposed to be easily inferred from the background preceding the conversation. However, not all of these propositions will be shared by the participants, owing to their differing perspectives and knowledge. Therefore, only a part of these propositions are mutually shared by the participants, forming their *common ground* [42]. As the conversation proceeds, new propositions may be added while existing ones are destroyed, and in so doing, the common ground *accumulates*. For example, if Alice says to Bob "I can't meet today, but I'm free tomorrow.", Bob understands that his assumption that he'd be meeting Alice today doesn't hold true anymore. At the same time, he concludes that tomorrow might be a good day to schedule a meeting, thereby adding a new proposition to the common ground. However, merely stating something is not enough to contribute to the common ground. The participants must actively work to ensure that each of them understands the content being added to the common ground. Therefore, when Bob says, "Let's meet tomorrow then.", Alice will be satisfied that Bob understood her statement about being free tomorrow as a possible agreement to meeting tomorrow, and will reply, "Yes, let's do that.", and now both Bob and Alice have a new proposition added to the common ground i.e. the mutual promise to meet tomorrow. The collective process by which the participants ensure the arrival at this mutual understanding of the information being accumulated to the common ground is called *grounding* [42].

Augmenting presuppositions of the participants is but one of the ways of grounding a conversation. Human cognition is believed to be *situated* [13] in its environment, meaning that cognition arises due to the interaction of an agent with its environment, and is inseparably coupled with the current context [12]. It is perhaps unsurprising, therefore, that the process of grounding language benefits from the context provided by the percepts available to the agent. For example, when Alice says to Bob, "Could you pass me a cup?", Bob must be able to locate a cup in his environment by looking for it. If he cannot find one, he is essentially unable to ground the utterance "cup" to any of the objects he can see. Therefore, Bob concludes that Alice's directions are misleading, and he asks Alice for a clarification to repair the break in grounding.

To be able to ground an utterance, one must be able to understand the meaning of its words or phrases, and meaning is embodied [43]. That is, understanding meaning involves enacting internal mental simulations that involve perceptual and affective faculties at one's disposal. A growing body of research work backs this claim. [44] [45] found that verbs referring to specific effectors (*licking, picking*) activated regions of motor cortex in the brain that were also activated by the actual movement of the corresponding effector (tongue, fingers, respectively). [46] considered the time it took for people to process fictive motion sentences i.e. sentences like 'The road runs through the valley', which contain a motion verb (here, *run*) but do not imply actual motion, and they found that it took longer for people to process such sentences when the previously established context suggested factors that would slow motion (for example, an older protagonist, a difficult terrain, longer distances, etc).

Pustejovsky and Krishnaswamy [47] emphasize the role of simulation in human-computer interaction, highlighting the use of game engines to create a rendered analogue to the "mental simulations" mentioned previously. This is facilitated via VoxML [4], a modeling language to create representations of objects, their attributes, and events enacted over these objects that can be used to construct a 3D visualization from a linguistic utterance. The unit of meaning in a language is called a *lexeme* which is arrived at by disregarding inflectional endings of words [48]. For example, the words "running", "ran", "runs" all map to the same lexeme "run". The

collection of lexemes is called the *lexicon*. In VoxML, each lexeme has a representation that corresponds to the lexeme's semantic content as a visualization [49]. VoxML forms the scaffold used to link lexemes to these visual instantiations or the "visual object concepts" – the *voxemes*. Together, the voxemes constitute the *voxicon*. A single lexeme may map to multiple voxemes. For example, the lexeme **plate** can be visualized either as a **square plate** or a **round plate**, each forming a distinct voxeme. Conversely, a single voxeme may be referred to by multiple lexemes. For example, both **dish** and **saucer** can refer to the single voxeme **round plate**.

VoxML is rooted in Generative Lexicon Theory [35], which decomposes word meaning in terms of Qualia relations (also called Qualia roles). The word Qualia (singular Quale) comes from Latin meaning "of what kind of thing". Qualia roles for a word represent the way the word is used in language that is attributed to the "world knowledge" gathered as part of human experiences. For example, the word "sit" is considered a Quale of the word "chair" because we know from our experience that one "sits" in a "chair". Similarly, the word "bake" is a Quale of the word "bread" as one "bakes" the "bread". In fact, "bake" and "sit" fulfill different qualia roles for "bread" and "chair", respectively, as suggested by the following division of qualia roles:

Formal/Atomic Conceptual superclass of the lexical item (`is-a` relation)

Constitutive/Subatomic Internal constitution of the entity (`part-of` or `made-of` relation)

Telic Typical purpose and function of the entity (`used-for` or `functions-as` relation)

Agentative How the object came into being i.e. its origin (`created-by` relation)

For example, a "book" `is-a` "publication" and it is `made-of` "text". A book `is-created` when an author "writes" it. The author's hope is that his "book" `is-used` for "reading" by someone.

The telic role of a lexical item can only be achieved under specific circumstances. This situational context is encoded as the object's *habitat* [50]. This along with description of what can be done to the object – its *affordances* – form the linguistic "dark matter" essential to visual instantiation of an object [5]. For example, a cup can afford being drunk from, being held, etc. So,

drinking from and holding are affordances of a cup. Habitats condition these affordances. For example, the cup must be full in order to be drank from. Affordances, in turn, can be Gibsonian or Telic. Gibsonian affordances are afforded by an object by virtue of its geometry. For example, grasping, lifting, etc. Telic affordances, on the other hand, are higher level affordances that exist by virtue of object's purpose or function. Telic affordances may be accomplished using Gibsonian affordances. As an example, reading is a telic affordance of a book.

A VoxML entity is one of `OBJECT`, `PROGRAM`, `ATTRIBUTE`, `RELATION`, and `FUNCTION`. Each of these, along with their structure, is described next.

OBJECT Model for nouns.

- **LEX** Lexical information of the object.
  - **PRED** Predicate lexeme denoting the object.
  - **TYPE** Object's type according to Generative Lexicon [35].
- **TYPE** Describes object geometry.
  - **ROTATSYM** Rotational symmetry across X, Y and Z axes.
  - **REFLECTSYM** Reflectional symmetry across planes XY, XZ and YZ.
- **HABITAT** Specifies object's habitats.
  - **INTRINSIC** Habitats intrinsic to the object, regardless of action it participates in.
  - **EXTRINSIC** Habitats that must be satisfied for particular actions to take place.
- **AFFORD_STR** Set of actions along with their requisite conditions that the object may take part in.
  - **GIBSONIAN** Low-level affordances by virtue of object's geometry.
  - **TELIC** High-level affordances by virtue of object's purpose or function.
- **EMBODIMENT** Embodiment of the object relative to an in-world agent (usually human).
  - **SCALE** Qualitative description of object's scale compared to the agent.

MOVABLE Whether the object is movable by the agent or not.

PROGRAM Verbs modeled as n-ary predicates that can take objects or other evaluated predicates as argument.

LEX lexical information of the program

PRED Predicate lexeme denoting the program.

TYPE Program's type according to Generative Lexicon [35].

TYPE Describes how the visualization of the action is realized.

HEAD Base form - state, process, transition, assignment, test (conditions that must hold during a transition).

ARGS Participants of the verb.

BODY Subevents executed as part of the program execution.

ATTRIBUTE Describe adjectival modifications.

LEX Lexical information of the attribute.

PRED Predicate lexeme denoting the attribute.

TYPE Scale and transitive property of the attribute.

SCALE Scale of the set of related attributes like nominal, binary, ordinal, interval, and ratio.

ARITY Whether the attribute requires comparison to other objects (transitive) e.g. small or not (intransitive) e.g. red.

ARG Variable representing the object to which the attribute is applied.

RELATION Relations specifying configuration information about two or more objects.

LEX Lexical information of the relation.

PRED Predicate lexeme denoting the relation.

TYPE Typing information associated with the relation.

CLASS Nature of the relation: configuration/force_dynamic.

VALUE Region connection calculus configuration relations.

ARGS Arguments participating in the relation represented as typed variables.

FUNCTION Predicates which take objects as argument and evaluate to geometrical regions.

LEX Lexical information of the function.

PRED Lexeme denoting the function.

TYPE Typing information associated with the function.

ARG Object being computed over.

REFERENT Any sub-parameters of **ARG** semantically salient to the function.

MAPPING Type of transformation the function performs over the object.

ORIENTATION Describes the space and primary axis of the function operates with.

SPACE function is performed in either world or object space.

AXIS primary axis and direction the function exploits.

ARITY Transitive or intransitive (similar to **ATTRIBUTE**s).

2.4 Human Activity Recognition

The goal of a human activity recognition system (HAR) is identifying actions or activities done by a person or a group of people [51]. Most of the human activity recognition focuses on analyzing video sequences. In particular, space-time approaches [52] focus on recognizing activities based on space-time features or on trajectory. Optical Flow has been used in many such approaches. Recent approaches have exploited 3D depth data and motion capture systems, partly enabled by the widely available, cheaper depth sensors like Microsoft Kinect. Such

shape-based methods focus on analyzing human silhouette represented as limbs jointly connected to each other. Recent approaches to this problem are described in more detail in [52].

2.4.1 EGGNOG

EGGNOG [7] (**E**licited **G**iant **G**allery of **N**aturally **O**ccurring **G**estures) is a multi-modal, continuous dataset of naturally occurring gestures. Gestures are performed by the participants in a spontaneous manner within the context of a shared physical task. The task involves collaboration between a participant pair so that the participants are separated in space, each facing a table and a TV screen on which they can view the other participant. One of the participants, the *signaler*, can also view a block pattern that the other participant, the *actor*, cannot. The task consists of the signaler instructing the actor as to how to recreate the block pattern through the use of gestures and sometimes, spoken language. The signaler is not limited by a pre-taught gesture vocabulary and is free to choose any gesture that best captures the move to recreate the block pattern.

The tasks are captured in various modalities for both the signaler and the actor. These modalities include RGB video, depth video, and skeleton data recorded by Microsoft Kinect v2 [53] devices. To aid in the labeling process, the trials are segmented into motions by detecting the local curvature maxima of the curves formed by high-dimensional body position data according to the algorithm described in [9]. Each motion, then, is manually labeled by an appropriate gesture label using a labeling tool called EASEL [8]. Of course, not all motions are meaningful gestures, making this a continuous dataset.

The gesture labels follow the format of body part: description. The body part may be one of 'body', 'head', 'right/left arm', 'right/left hand'. The description provides additional information about the gesture like pose, direction, etc. For example, right hand: claw, down describes the claw gesture made by the right hand such that the palm is facing down towards the table.

In addition to the gesture labels, the dataset also provides intent labels. Intents record the inferred meaning of the signaler's instructions within the context of communicating the block

pattern to the actor and may span multiple gestures and words. For example, an instruction that involves the signaler holding the hand completely open with fingers completely closed such that the palm is facing to the left, followed by a sweep of the arm in the same direction may be associated with the intent of sliding a block to the left. Many more intents like stack, row, column, here, there, etc. are recorded in the dataset.

2.4.2 Gesture Recognition

Gestures are one of the more intuitive ways to interact with machines. Gesture recognition refers to the whole process of tracking human gestures to their representation and conversion to semantically meaningful commands [54]. Vision based methods for gesture recognition employ one or more cameras and analyze video sequences to identify gestures. Although such methods are more convenient and comfortable to use, they suffer from configuration complexity, high computations cost, and occlusion problems. Availability of depth sensors has enabled another approach called depth-based gesture recognition. Depth sensors make hand segmentation, one of the essential tasks in gesture recognition, trivial. Cheng et al. [55] summarizes recent approaches to gesture recognition problem. Deep learning techniques have been applied with great success as well, surpassing many of the previous state of the art results. In particular, Recurrent Neural Networks (RNNs), Long Short-term Memory (LSTMs), and 3D Convolutional Nets have made fast and reliable gesture recognition systems possible. Readers can refer to [56] for more details on deep learning based approaches to gesture recognition.

2.5 Embodied Agents

Embodied conversational agents (ECAs) are embodied agents living in a virtual world that are capable of engaging in conversation with one another and with humans employing the same verbal and nonverbal means that humans do. Creating believable ECAs is a major undertaking requiring knowledge from multiple disciplines like computer animation, modeling, psychology, etc. ECAs need to have a realistic appearance, have flexible motion modeling,

and intelligent behavior modeling [57]. ECAs have been developed for military rehearsal exercises [58], distress evaluation [59], etc. among other uses.

The Mission Rehearsal Exercise system [58] aims to inculcate leadership skills to deal with high-stakes situation in an effective way. The system puts the trainee in a virtual environment where he is on his way to assist his fellow soldiers when he discovers that one of the vehicles in his platoon has been involved in an accident with a civilian. While the TV camera crew gathers, the trainee has to make decisions under stress that will impact the final outcome. To make the situations as life-like as possible, the virtual humans possess an integrated set of abilities such as the ability to perceive and act in the virtual world, engage in face to face dialogues, and display realistic emotions. To ensure flexibility, a blackboard architecture (Section 2.2) was adopted that allowed individual components to access shared intermediate results of other components. Emotions were modeled based on appraisal theory.

SimSensei Kiosk [59] system consists of a virtual human interviewer, Ellie, who is designed to engage in face to face conversations and can automatically assess verbal and non-verbal indicators of anxiety, depression, etc. The developers aggregated data from human-human interactions in the context of distress assessment, and followed it up with a Wizard of Oz experiment to select an adequate set of utterances and non-verbal behavior to differentiate between distressed and normal interviewees. The perception abilities of the system consists of a multimodal system that allows real-time synchronized capture of multiple modalities and their fusion. The modalities included video, audio, head position, gaze, face expression, etc. For animating Ellie, Smart Body character animation was used.

2.6 Pointing

A pointing gesture is "a communicative body movement that projects a vector from a body part to indicate certain direction, location or object" [60]. It can also be used to trace a path or a shape. Pointing is one of the most common communicative signals in our day to day life, and is one of the earliest acquired communicative ability by humans [60].

Being such an essential part of human-human communication, various techniques have been proposed over the years for selecting objects in virtual environments. One of the most fundamental tasks performed by users in GUIs is object selection [61]. People often interact with large digital displays by pointing from a distance. Ray-pointing is a convenient selection technique to this end. By using the intersection of the virtual ray with a surface, the location being pointed at is determined [62]. It is easy to use, allows multiple people to interact with the same display without their bodies getting in the way, and does not require a physical surface to operate (as opposed to a mouse pointing device). However, like any ray-casting technique, it suffers from lack of precision in selecting distant objects. This is because distant objects occupy proportionally lesser surface of the display, thereby demanding greater precision from the user. Moreover, the effect of hand jitter amplifies with distance, further deteriorating precision [63]. [62] recognize four variations of ray pointing techniques as follows:

Regular Laser Pointing The user holds a physical device such that the position and orientation of the device determines the direction of ray. The device need not be an actual laser as long as the computer system can recognize the intersection of ray with the display. It is also called virtual pointing.

Arrow Pointing It is the same as regular laser pointing except that the pointing device position is constrained to be aligned with the user's eye, much like one would do when playing a game of darts.

Image-Plane Pointing The virtual ray is determined as originating from user's eye to another point that can be controlled by the user, usually the tip of their thumb or a pointing device. This technique requires tracking of user's eye in some way.

Fixed-Origin Pointing In image-plane pointing, one of the points was fixed at the user's eye. By relaxing this restriction, one can fix the origin of the ray to be any point in space while the user controls the other point. This is called Fixed-origin pointing.

Chapter 3

System

This chapter describes the complete system in detail by going over each of its components. We will first describe the Virtual Environment – the environment housing Diana and much of her manipulable universe. Next, we peel the layers off the mechanisms behind Diana's abilities by describing the Perception modules that represent Diana's perceptual faculties, Cognitive modules that control Diana's decision making and Behavior modules that drive Diana's motor abilities. All these components are centralized in an asynchronous datastore called the blackboard which allows a real time synchronization between these multiple independent modules (see Figure 3.1), thereby making Diana the sophisticated multimodal agent that she is.

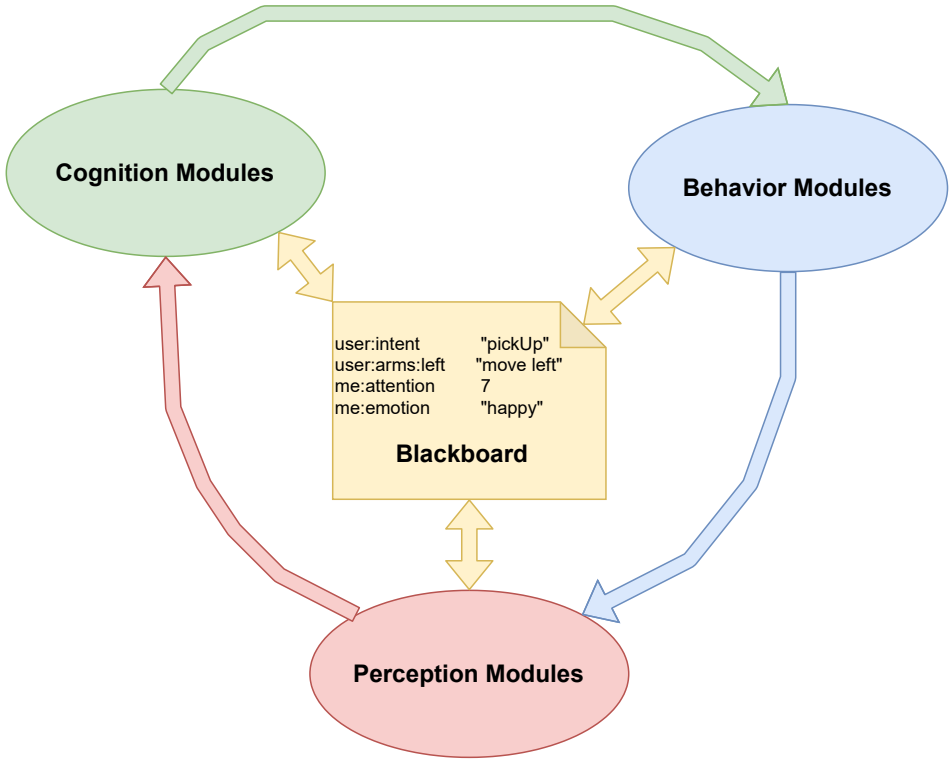
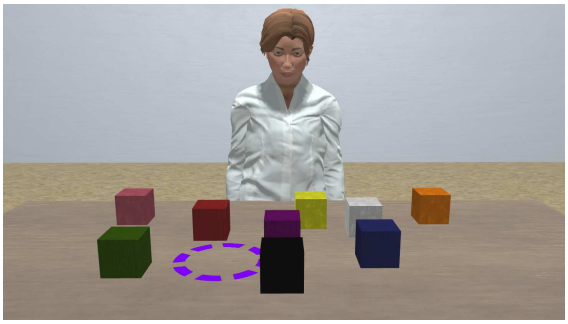


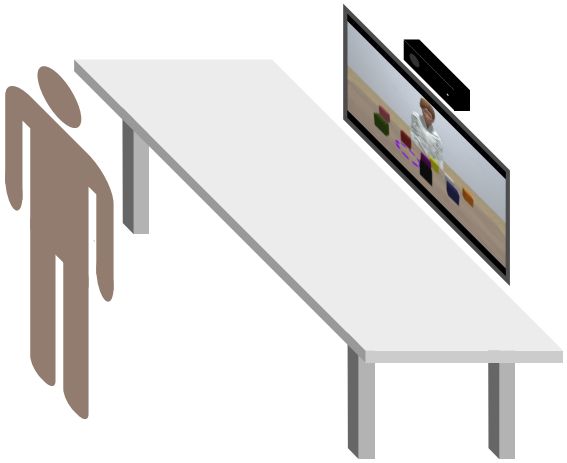
Figure 3.1: Action Perception Cycle [1]

3.1 Virtual Environment

The virtual environment is created in Unity 2018. The environment consists of a table with blocks placed on it, essentially the blocks world (Figure 3.2a). The agent stands on the opposite side of the table as the camera pointing direction. The only objects that can be manipulated are the blocks on the table. These can be moved around with an appropriate command from the interacting user (Figure 3.2b).



(a) Agent as visible to user



(b) User interacting with the system

Figure 3.2: Virtual Environment

The agent is called Diana. She is modeled using Adobe Fusion software. The model comes with blend shapes that allow us to manipulate her facial expressions. Her motor abilities are implemented using a combination of animations and Inverse Kinematics. We use one wave animation and four reaching animations, all sourced from Carnegie Mellon Motion Capture Database [64]. Additionally, four cross-reaching animations were recorded separately with Microsoft Kinect, and then cleaned with Autodesk Maya. With a total of eight reaching animations, we use animation blending capabilities in Unity to create intermediate reaching animations that allow Diana to reach any part of the table.

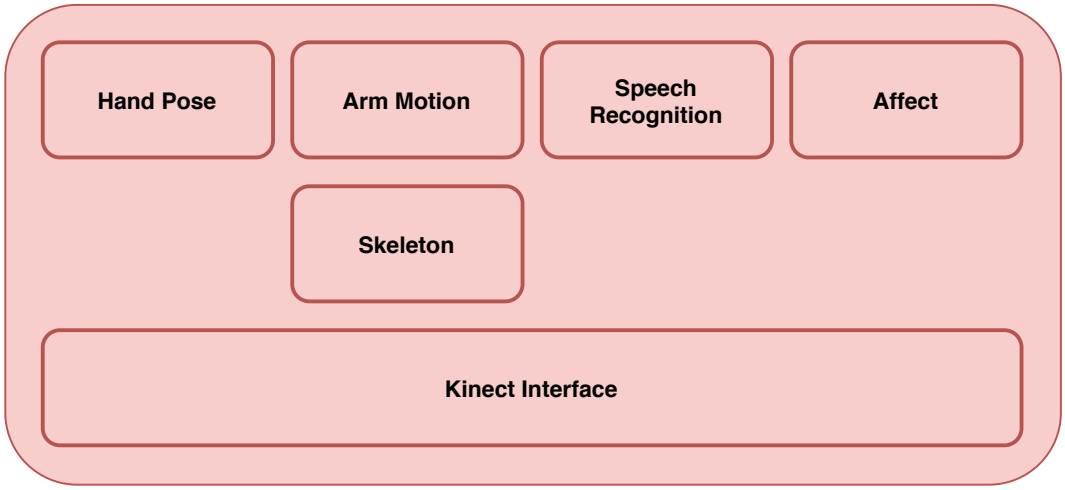


Figure 3.3: Perception Modules

3.2 Perception Modules

The perception modules are responsible for providing the sensory input to the system. The data is written to the blackboard to be used by the other components of the system. Broadly, we have six perception modules:

- Kinect Interface
- Skeleton
- Hand Pose
- Arm Motion
- Speech Recognition
- Affect Recognition

3.2.1 Kinect Interface

The Kinect for Windows SDK 2.0 exposes the Kinect v2 programming API for C# projects on Windows operating system. To meet performance requirements of our system and minimize system complexity, it was imperative to enable the Unity C# scripting environment to access

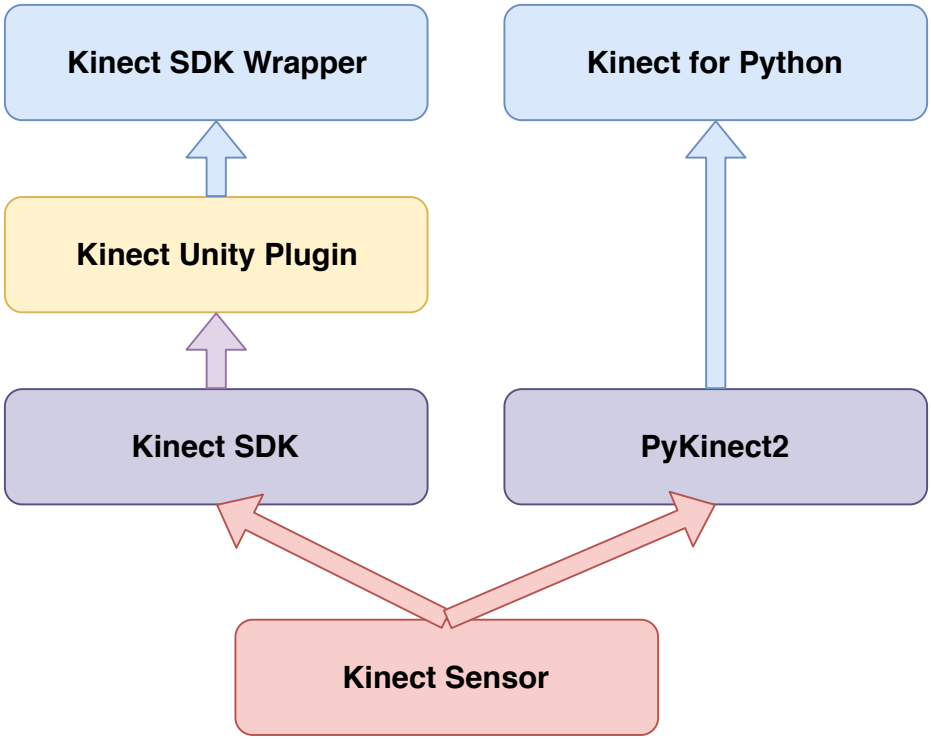


Figure 3.4: Kinect Interface

the Kinect v2 API as directly as possible. This is accomplished via the *Kinect Interface*. The Kinect Interface utilizes Kinect v2 Unity plugin [65] released by Microsoft to access Kinect v2 API directly in Unity. It further abstracts the most common operations provided by the Kinect v2 API into Kinect SDK Wrapper. The wrapper ensures proper resource management and increases code manageability. Any Unity C# script can access this wrapper for high-level interactions, but retains the flexibility to directly access the Kinect v2 API if needed.

The Kinect Interface consists of the following four layers, listed in order of increasing abstraction:

- Kinect Sensor
- Kinect for Windows SDK 2.0
- Kinect v2 Unity plugin
- Kinect SDK Wrapper

Kinect Sensor

Kinect is a motion-sensing input device developed by Microsoft. It has been through four different iterations over the years:

- Kinect for Xbox 360 (2010)
- Kinect for Windows (2012)
- Kinect for Xbox One (2013)
- Azure Kinect (2020)

We use Kinect for Xbox One, also called Kinect for Windows v2, is used for our setup. Henceforth, it is described simply as *Kinect v2*. Kinect v2 consists of three sensors: a color camera, an infrared camera, and an array of microphones.

The Kinect v2 color camera captures 1080p video. Its field of view is 84.1 × 53.8 degrees. With its wide field of vision, the sensor can detect a user up to 3 feet from the sensor compared to six feet for the original Kinect.

The Kinect v2 infrared camera has a resolution of 512×424px, which is three times compared to its predecessor. Its field of view is 70.6 × 60 degrees. It measures depth of the scene objects based on the time-of-flight measurement principle. Essentially, the infrared light is cast from the sensor. When it gets reflected by the obstacles in the environment, the time of flight is registered for each pixel. Internally, wave modulation and phase detection is used to estimate the distance to obstacles (indirect ToF) [66]. The working range for the depth sensor is specified as 0.5m to 4.5m.

The Kinect uses a body tracking algorithm to identify the position and orientation of 25 individual joints (including thumbs). It can simultaneously track joints for up to six users. It also provides detection of face attributes for each detected person. For e.g., you can know if the user is looking at the camera or not (Engaged), expressions (happy or neutral), appearance (if wearing glasses, the skin color or hair) and activities (eyes open or closed, mouth open or closed).

Kinect SDK

The Kinect v2 provides a programming interface via the Kinect for Windows SDK v2.0. The SDK provides a multitude of functions and classes that provide access to all the sensor data. These are described as follows:

Color Provide access to the HD resolution RGB color frame.

Depth Provide access to the depth frame.

Body Provide access to the tracked bodies and the 25 joints' position and rotation for each tracked body.

Infrared Provide access to the infrared image of the scene.

Face Detect user's face's keypoints such as eyes, mouth, etc. and facial expressions.

FaceHD Detect user's skin color, hair, and face mesh points.

Coordinate Mapping Map the coordinates in image acquired with one sensor to an image acquired with another sensor. For example, to map a joint position calculated with body tracking to a pixel in the color image, you would use this.

Audio The Kinect uses an array of microphones that can be accessed to get directional audio. Additionally, it can also associate a generated sound to a tracked body.

Kinect Unity Plugin

The Kinect SDK v2.0 is provided for development in Unity game engine through a Unity Package created by Microsoft [65]. The plugin wraps all the core Kinect functionality, and face detection APIs in this plugin. Since the majority of our virtual environment is implemented in Unity, we use the Unity plugin as an intermediate for our Kinect Interface.

Kinect SDK Wrapper

Although the Kinect SDK is comprehensive, it can be very verbose to work with. So, to remedy this, we develop a wrapper around Kinect SDK API. Our wrapper is composed of three classes:

- `KinectSensor` This class wraps the Color, Depth, Body, Infrared, Face, Coordinate Mapping, and Audio APIs into a uniform interface. The primary advantage of having such a class is that it takes care of closing access to individual sensors so as to avoid memory leaks. Additionally, it synchronizes all frames using a 64-bit integral timestamp that represents the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight, January 1, 0001.

- `KinectSensor.MultiSourceFrameArrived` This event is triggered when the synchronized Color, Depth, Infrared, Body frames are generated.

- `KinectSensor.FaceFrameArrived` This event is triggered when the Face frame is generated.

- `KinectSensor.AudioBeamFrameArrived` This event is triggered when the Audio frame is generated.

- `KinectEventArgs` The information associated with each of the three events mentioned above is encapsulated in the following classes:

  - `MultiSourceFrameArrivedEventArgs`
  - `FaceFrameArrivedEventArgs`
  - `AudioBeamFrameArrivedEventArgs`

- `KinectException` Any failure that occurs in our Kinect SDK wrapper is encapsulated inside `KinectException` with the actual exception stored as a field for finer debugging.

Kinect for Python

Some of the perception modules do not live inside the Unity environment, and are separate Python processes. In order to avoid throttling the network traffic and disk space, we use the PyKinect2 library [67], a wrapper library that exposes Kinect for Windows v2 API in Python. The Kinect for Python component works alongside the Kinect SDK wrapper, and provides access to the same Kinect sensor.

3.2.2 Skeleton

The Skeleton module is responsible for extracting the user joint information provided by Kinect Interface and writing it to the blackboard. The joint information consists of the position and the orientation of the 25 tracked joints of the user nearest to the setup. The distance of the user from the setup is measured as Euclidean distance of the user's spine base joint from the Kinect sensor. The position is represented relative to Kinect origin and each component is specified in meters. The orientation is represented as a quaternion.

3.2.3 Hand Pose

The Hand Pose module is started as an external Python process from within Unity. The module accepts the depth frame from Kinect using PyKinect2 [67] library. It also retrieves the hand wrist joint coordinates in the depth frame. Then, it proceeds to crop the depth frame around these joint positions to generate two hand depth frames. Finally, the two depth frames are pre-processed and passed into ResNet-style deep convolutional neural network (DCNN) [6] to produce hand gestures. A total of 34 gestures can be recognized, which includes hand gestures like thumbs up, thumbs down, counting (1-5), pointing, etc.

3.2.4 Arm Motion

The Arm Motion module, like the Hand Pose module, is also started as an external Python process from within Unity. The module accepts the body frame from Kinect for Python wrapper. The body frame consists of the joint positions of the user closest to the sensor. This data

is passed into an LSTM recurrent neural network [20] to classify the arm motions. The arm motions specify which of the six directions the arm moved in.

3.2.5 Speech Recognition

The speech recognition is performed using an off-the-shelf automatic speech recognition service, namely Google Cloud Speech-to-text [68]. To access the API from within Unity, an open-source Unity plugin named UnityGoogleStreamingSpeechToText [69] is used.

3.2.6 Affect Recognition

The Affect Recognition module is built on top of Affectiva's [70] Affdex SDK for emotion measurement. It can analyze spontaneous facial expressions using on-device processing power. The emotion metrics that can be detected using this SDK include anger, contempt, disgust, fear, joy, sadness and surprise, with reported accuracy in 90^{th} percentile [71]. It can also detect a few other non-emotion metrics like age, gender, ethnicity, etc. The underlying classifiers are trained on a massive dataset with more than 6.5 million faces from 87 countries. However, the SDK is no longer directly available to developers.

The Affect Recognition module accepts the Color frame from Kinect Interface and passes it along to the Affdex SDK, which generates the dominant emotion. The dominant emotion is then relayed to the blackboard, at which point it becomes visible to all the modules in the system.

3.3 Blackboard

It is necessary to use an architecture that supports asynchronous coordination between multiple processes running in real time. With this in mind, we opted for a layered architecture of independent modules such that the coordination between all the modules, each providing expertise for a very specific area, is enabled by the blackboard [72]. The blackboard is the central data store that all modules and the virtual environment interact with. Each module can

read and write to the blackboard independently of any other module. In turn, a module can take a decision based on the "expertise" shared by all other modules on the blackboard.

Externally, he blackboard is essentially a key-value store. The keys are hierarchical, with each level separated by a : (colon). For example, `user:hands:left` describe the left hand's gesture of the user. Internally, the blackboard is implemented as a mapping of keys to the subscribers to that key. The modules that rely on a particular key will subscribe to the changes to that key. When the key changes, each subscribed module is notified, and can react accordingly.

To ensure that blackboard is available to external processes, it provides a socket API to read or write any of its keys. The socket API is available in both Python and C# languages. For example, the Hand Pose module and the Arm Motions module, both of which run as external Python processes, rely on the Python version of socket API to to write the detected hand gesture and arm motion to the blackboard, respectively. Similarly, the C# modules like Attention module use the C# version of the socket API to write the attention value to the blackboard.

3.4 Cognition Modules

Cognition modules represent the thinking processes of Diana. These work with the sensory inputs provided by the perception modules, and transform those into an understanding of the environment - both physical and virtual. The physical environment primarily constitutes understanding the actions performed by user, or the state of his/her participation. The virtual environment understanding is manifested by extrapolating user's pointing to locate an object in the virtual environment. The different cognition modules are described below:

User Pointing The User Pointing module extrapolates the line joining the user's shoulder and wrist joint into the virtual environment, and the point of intersection of this line on the virtual table is determined as the pointing location in the virtual environment.

Engagement The Engagement module determines if the user is engaged or not. The user is said to be engaged when he/she is standing in the engagement zone.

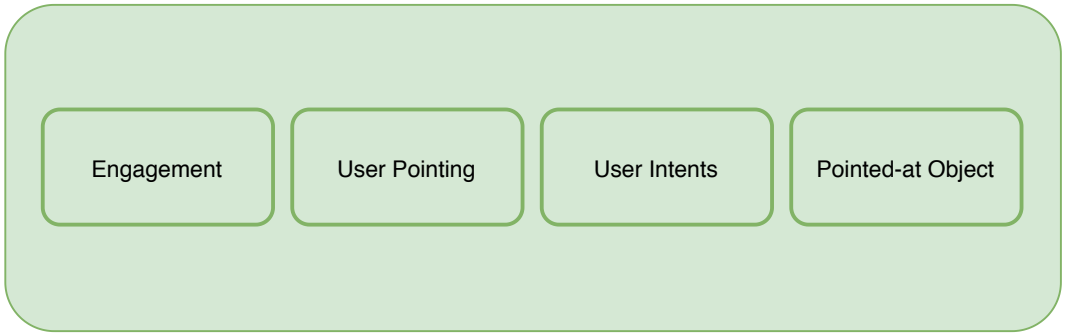


Figure 3.5: Cognition Modules

User Intents The user intents describe what the user intends the agent to do using non-verbal behavior. They are determined from the hand pose and the arm motions.

Pointed-at Object The pointed-at object is determined by overlapping a sphere centered at the pointed-at location with the manipulable objects in the scene. If the same object remains the focus of pointing for a given amount of time, then it is set as the pointed-at object.

3.4.1 State Machines

The user intents are determined using Finite State Machines. However, the transition rules are not just based on the input, but can be augmented with time-constraints. For example, to detect if the user did a positive acknowledgment, he/she needs to keep the hand in a thumbs up gesture for a certain amount of time. This helps in smoothing out the noise in the user's input.

3.4.2 User Intents

When the user does a gesture with hands and/or arm motions, they may be interpreted as a signal to the agent to perform an action. We call these user intents. There are different user intents that can be understood by the agent:

Positive Acknowledgment User does a thumbs up with either hand.

Never mind User raises either hand with the palm facing the agent, like a stop sign.

Push *direction* User does a push gesture by swiping the arm in left, right, front or back directions with the hand palm facing the direction of the swipe.

Negative Acknowledgment User does a thumbs down with either hand.

Wave User waves with either hand.

Wait User does a count-one gesture with either hand, as if gesturing "wait a second".

Claw User does a claw gesture such that the palm is facing down.

3.5 Behavior Modules

Behavior modules alter the state of the virtual environment. The change could result in the agent acting on the environment in some way, or it could be entirely limited to modifying the internal state of the agent. An example of the former would be Emotion module, while the latter case is illustrated by the Attention module when the attention level isn't high enough.

Alternatively, the change could just modify the virtual environment without changing the agent's state at all. A great example of a behavior module working this way is the User Pointing module. The user pointing module highlights the estimated location being pointed at by the user as a circle projected on the virtual table, and in doing so, only alters the virtual environment state without prompting any reaction from the agent (Diana). Admittedly, altering virtual environment state with a rendering like this, which has no analogue in the real world, shifts the system away from reality. However, as we learned during user trials, it is a necessary concession to enhance users' experience with the system because without it, users find it really hard to ground their deictic references.

User Pointing The user pointing module is responsible for highlighting the pointed-at location in the virtual environment. This is essential for providing user with the feedback of his pointing gesture. The module draws a sprite in the shape of a dashed circle on the virtual table surface, and keeps the dashed circle rotating at a constant speed as long as the user

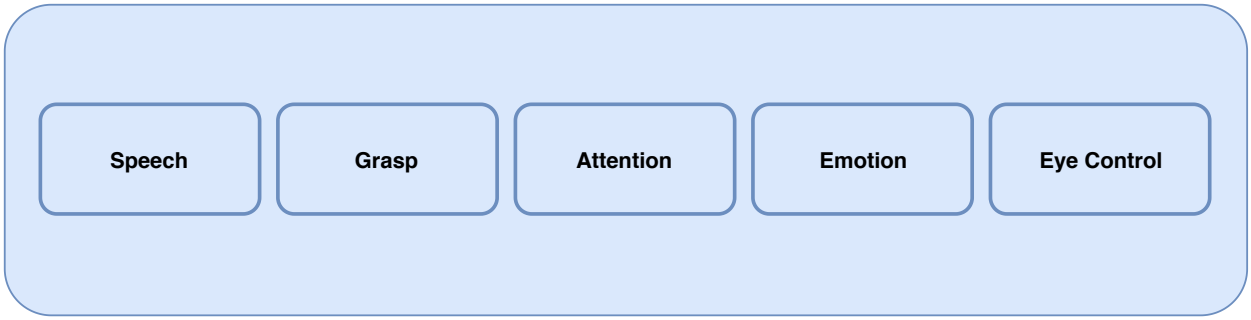


Figure 3.6: Behavior Modules

is adjusting the pointing by moving the arm. Once the user fixes the pointed at location by keeping the arm still for a short time, the circle stops rotating.

Eye Control This module controls the eyes, based on inputs like alertness and attention, as well as internal state such as the need to blink. One important finding from our experience with the system that allowing Diana to follow the object she is acting on with her gaze

Speech Output This module detects an intent to speak the given text, and performs text-to-speech (TTS) output. The TTS functionality is provided by RT-Voice Pro asset available in the Unity Asset Store.

Attention This module controls Diana's attention. For example, when the user is pointing at the screen or when the user is in the engagement zone of the setup, Diana's alertness level will increase. If the alertness level is high enough, she will respond by directing her gaze at the user.

Emotion This module controls Diana's facial expressions and is affected by her perception of user's emotions and the ongoing task. For example, she will smile when greeting a user. Similarly, she will express concentration when the user is describing a task to perform.

Grasp The grasp module is a low-level motor control module that directs the agent's actions. The available actions include: "reach" to reach a given location on the plane of the table as if to grab it or indicate a prospective destination location, "move" to move the hand

to a different 3D location, "hold" to hold the object at the reached location, "release" to release the held object, and finally the "unreach" action to retract the hand to an idle pose.

3.6 Motor Control System

This chapter describes how Diana's motions are controlled, which is critical to the working of Diana's motor control system. We'll start with a description of Unity's animation system. Then, we'll provide the concrete implementation details relevant to the system.

3.6.1 Unity Animations

Animations in Unity can be of two types: External and Internal.

External Animations

The external animations are recorded outside Unity Editor. These could be humanoid animations that are recorded in a Motion Capture studio. Alternatively, these could be animations created in an animation software such as Autodesk 3ds Max. External animations cannot be edited within Unity.

Internal Animations

Unity allows creating and editing of animations within the Unity Editor as well. This is accomplished by setting key points in a timeline. A key point describes a property's value at a certain point in time. As far as animation is concerned, any component property whether Unity provided or user written can be animated like this. For example, all Game Objects have a Transform component with Position, Rotation, and Scale properties. This means you can animate all three properties for any Unity game object. Similarly, if the user wrote a script (as a `Monobehavior` component, any public field becomes the component's property, and hence is also available to the Unity animation system to be animated.

Animator Controller

An animator controller is a component that allows a user to describe a state machine such that each state has an animation associated with it as well as transitions originating out of it. The transitions are ordered, such that each transition's conditions are evaluated only if the conditions associated to the transition above it are not true. The transitions are controlled by Animation parameters and conditions. The animation parameters are controlled via user scripts, as these represent conditions that must evaluate to true before the animation is executed. There are 4 types of animation parameters:

1. Boolean: A true/false flag. Available conditions are that the parameter must be true or false.

2. Trigger: Triggers can only be set, but are implicitly unset the first time they are evaluated. Only available condition is that the trigger must be set.

3. Float: Floating point numbers. Available conditions are greater than or less than checks.

4. Integer: Integer numbers. Available conditions are greater than, less than, equal to and not equal to checks.

Interruptible Transitions

Many of Diana's motor abilities are implemented using blended animations guided by Inverse Kinematics. However, by default, all animation transitions are uninterruptible. Therefore, as soon as all the conditions related to animator parameters of a transition are true, the transition will take place, even if the next instant all the conditions evaluate to false. This will result in a noticeably slow response to the user. For example, imagine the user says "stop" while Diana is reaching out to grab a block. If not for interruptible animations, Diana would complete the animation to reach the block, and only then accept events to stop the motion. This would result in very unrealistic behavior since humans are quick to respond to such signals. Moreover, say the user changed his/her mind midway while selecting one block and then suddenly choosing

another block. This would also require Diana to react as soon as a different course of action is requested by the user.

Thankfully, Unity allows an ongoing transition to be interrupted by allowing the developer to define transition order. The transitions higher in this order can interrupt a transition lower in the order.

1. **Current State** Current state is the state that the transition is originating from. This order setting allows a transition from the current state to interrupt an ongoing transition.

2. **Next State** Next state is the state that the transition is set towards. The order setting allows a transition in the next state to interrupt the ongoing transition.

3. **Current State then Next State** The transitions out of the current state are checked for interruption first. If no interruption is found, the transitions out of the next state are checked next.

4. **Next State then Current State** The transitions out of the next state are checked for interruption first. If no interruption is found, the transitions out of the current state are checked next.

Given the choice of one of the above, all transitions are evaluated in the order they are specified even when one of the transitions is ongoing. If any of the transitions evaluates to true, then the ongoing transition is interrupted, and the new transition takes over.

Animation Blending

Not all motions can be described by a finite set of pre-recorded animations. Animation blending allows creation of intermediate animations that can be finely controlled using animation parameters, mostly floating type. The animation blending is accomplished by using a Blend Tree state in the Animator Controller.

Unlike the usual Animator Controller state that only has one animation associate with it, a Blend Tree state has multiple animations attached to it. Each animation in the Blend Tree is

also parameterized by one or more animation parameter values. When animation parameters have a value that is not the exact same as set in the blend tree for an animation, an intermediate animation is generated.

3.7 User Interaction Scenarios

The following sections explain some of the common use cases. Each section first describes the sequence of events as it would be perceived by the user. It then proceeds to explain all the intermediate steps that the system follows highlighting the communication and interplay between different system modules. The following use cases are discussed:

1. Engagement
2. Mutual readiness for dialogue
3. User pointing
4. Grabbing a block
5. Vertically stacking a block
6. Horizontally stacking a block
7. Interruption

3.7.1 Engagement

The engagement scenario consists of the following steps:

1. User steps into the engagement zone.
2. Diana says "Hello."

Under the hood, the following happens:

1. User steps into the engagement zone.

2. The Kinect Interface captures the sensor data from Kinect device.

3. The Skeleton Perception module extracts the joint data for the user from the sensor data, and writes it to the blackboard.

4. The Engagement Cognition module reads the position of the spine joint from the blackboard, and verifies that it falls within the bounds set for the engagement zone.

5. The Engagement Cognition module sets the key `user:isEngaged` on the blackboard to `true`.

6. The Attention Cognition module detects the change to `user:isEngaged` and sets the alertness level to a high value on `me:alertness`.

7. The Eye Gaze Behavior module detects the change to `me:alertness` and responds by setting the gaze to be towards the user. It also makes Diana's eyes to be wide open.

8. The Dialogue Interaction Behavior module detects the update to `user:isEngaged` and sets the `me:speech:intent` to "Hello."

9. The Speech Output Behavior module detects the update to `me:speech:intent` and starts speech synthesis for the given text.

10. Diana says "Hello."

3.7.2 Mutual readiness for dialogue

This scenario assumes that the user is already engaged and consists of the following steps:

1. User waves.

2. Diana says "I'm ready to go."

Under the hood, the following happens:

1. User waves.

2. The Kinect Interface captures the sensor data from Kinect device.

3. The Skeleton Perception module extracts the joint data for the user from the sensor data, and writes it to the blackboard.

4. The Wave Intent Cognition module reads the positions of elbow and wrist joints, and verifies wave motion over time.

5. The Wave Intent Cognition module sets the key `user:intent:isWave` on the blackboard to `true`.

6. The Dialogue Interaction Behavior module detects `user:intent:isWave` key being updated and sets `me:speech:intent` to "I'm ready to go."

7. The Speech Output Behavior module detects the update to `me:speech:intent` and starts speech synthesis for the given text.

8. Diana says "I'm ready to go."

3.7.3 User pointing

This scenario assumes that the user is already engaged and ready, and consists of the following steps:

1. User points at the screen.

2. The pointing marker appears on the virtual table.

Under the hood, the following happens:

1. User points at the screen.

2. The Kinect Interface captures the sensor data from Kinect device.

3. The Skeleton Perception module extracts the joint data for the user from the sensor data, and writes it to the blackboard.

4. The User Pointing Cognition module reads the position of shoulder and wrist joints for both arms, and determines if the user is pointing. If the user is determined to be pointing, it sets `user:isPointing` to `true`, otherwise `false`.

5. If the user is pointing, the User Pointing Cognition module also uses the joint data to approximate a 2D location on the virtual table and writes these coordinates on the blackboard as `user:pointPos`. If the approximated 2D location is on the virtual table surface, `user:pointValid` is set to `true`, otherwise it is `false`.

6. If `user:pointPos` changes and `user:pointValid` is `true`, the Pointing Marker component draws the pointing marker at the updated location.

7. The pointing marker appears on the virtual table.

3.7.4 Grabbing a block

This scenario assumes that user is already engaged and ready, and consists of the following steps:

1. User points at the screen.

2. User guides the pointing marker to a virtual block.

3. User holds the pointing maker on the block for a short time.

4. Diana says "Ok" and proceeds to grab the block.

Under the hood, the following happens:

1. User points at the screen

2. All the steps listed in Section 3.7.3 are repeated in a loop until the pointing marker appears on the virtual block.

3. Simultaneously, the Pointing Intent Cognition module checks blocks for collisions in the vicinity of the `user:pointPos` location.

4. If there is consistently some block that passes collision check and it's the same block for some given time period, then the Pointing Intent Cognition module sets the given block's name to `user:lastPointedAt:name`. It also sets the given block's position to `user:lastPointedAt:position`. Otherwise, it sets both to empty values.

5. When `user:lastPointedAt:name` is set to a non-empty value, the Dialogue Interaction Cognition module sets the intended action to grab the block by assigning `"reach"` value to `me:intent:action`, the location of the block to `me:intent:target`, and the name of the target to `me:intent:targetName`.

6. Simultaneously, the `me:speech:intent` key is set to "Ok."

7. The Grasp Behavior module responds to the update to `me:intent:action` key by starting the reaching animation towards the location specified in `me:intent:target`.

8. Simultaneously, the Speech Output Behavior module reacts to the change in the value of `me:speech:intent` and starts speech synthesis for the given text value.

9. Diana says "Ok" and proceeds to grab the block.

3.7.5 Vertically stacking a block

This scenario assumes that user is already engaged and ready, and consists of the following steps:

1. User points at the screen.

2. User guides the pointing marker to a virtual block (say A) and holds.

3. Diana proceeds to grasp the block.

4. User points at the screen again.

5. User guides the pointing marker to a virtual block (say B) and holds.

6. Diana says "OK." and proceeds to put block A on top of block B.

Under the hood, the following happens:

1. User points at the screen.
2. All the steps listed in Section 3.7.3 are repeated in a loop until the pointing marker appears on the virtual block A.
3. All the steps listed in Section 3.7.4 are repeated so that Diana grasps the block A.
4. User points at the screen again.
5. All the steps listed in Section 3.7.3 are repeated in a loop until the pointing marker appears on the virtual block B.
6. The Dialogue Interaction Cognition module computes the path to reach the location on top of the block B i.e. the target location.
7. The Dialogue Interaction Cognition module then sets Diana's intended action to move the block to the target location by setting `me:intent:action` to `move` and setting `me:intent:target` to the target location vector.
8. Simultaneously, the `me:speech:intent` key is set to "Ok."
9. The Grasp Behavior module responds to the update to `me:intent:action` key by starting the moving animation towards the location specified in `me:intent:target`.
10. Simultaneously, the Speech Output Behavior module reacts to the change in the value of `me:speech:intent` and starts speech synthesis for the given text value.
11. Diana says "OK." and proceeds to put the block A on top of block B.

3.7.6 Horizontally stacking a block

This scenario assumes that user is already engaged and ready, and consists of the following steps:

1. User points at the screen.
2. User guides the pointing marker to a virtual block and holds.
3. Diana proceeds to grasp the block.
4. User does a "push" gesture.
5. Diana says "OK." and proceeds to slide the block against another block.

In the above, the "push" gesture could be either "push left" or "push right".

Under the hood, the following happens:

1. User points at the screen.
2. All the steps listed in Section 3.7.3 are repeated in a loop until the pointing marker appears on the virtual block.
3. All the steps listed in Section 3.7.4 are repeated so that Diana grasps the block.
4. User does a "push" gesture.
5. The Kinect Interface captures the sensor data from Kinect device.
6. The Hand Pose Perception module reads the Depth frames for both hands from the Kinect Interface, generates gestures for both hands, and writes the recognized gesture labels to `user:hands:left` and `user:hands:right` respectively.
7. The Skeleton Perception module extracts the joint data for the user from the sensor data, and writes it to the blackboard.

8. The Arm Motion Perception module reads the skeleton data and generates arm motion labels in `user:arms:left` and `user:arms:right` respectively.

9. The Push Intent Cognition module combines both `user:arms` and `user:hands` labels into a user push intent and sets `user:intent:isPushLeft` to `true` if the user intent is recognized to be "push left". Similarly, `user:intent:isPushRight` is set to `true` if the user intent is recognized as "push right".

10. When either `user:intent:isPushLeft` or `user:intent:isPushRight` is set to `true`, the Dialogue Interaction Cognition module identifies the block to slide against i.e. the target block, and computes the path to reach the location beside the target block i.e. the target location.

11. The Dialogue Interaction Cognition module then sets Diana's intended action to move the block to the target location by setting `me:intent:action` to `move` and setting `me:intent:target` to the target location vector.

12. Simultaneously, the `me:speech:intent` key is set to "Ok."

13. The Grasp Behavior module responds to the update to `me:intent:action` key by starting the moving animation towards the location specified in `me:intent:target`.

14. Simultaneously, the Speech Output Behavior module reacts to the change in the value of `me:speech:intent` and starts speech synthesis for the given text value.

15. Diana says "OK." and proceeds to slide the block against another block.

3.7.7 Interruption

This scenario assumes that user is already engaged and ready, and consists of the following steps:

1. User points at the screen.

2. User guides the pointing marker to a red colored virtual block and holds.

3. Diana proceeds to grasp the block, but before Diana has grasped it, user says "No, the blue one".

4. Diana seamlessly transitions into grasping the blue block.

Under the hood, the following happens:

1. User points at the screen.

2. All the steps listed in Section 3.7.3 are repeated in a loop until the pointing marker appears on the virtual block.

3. All the steps listed in Section 3.7.4 are repeated so that Diana begins to grasp the red block.

4. Before Diana has grasped the red block, user says "No, the blue one".

5. The Speech Recognition Perception module recognizes that the user is speaking and sets `user:isSpeaking` to `true`.

6. When the user is done speaking, the Speech Recognition Perception module sets the speech transcription on the blackboard as `user:speech`.

7. The NLU Cognition module reacts to the change in `user:speech` and sets the intended action to instead grasp the blue block in the middle of ongoing grasp action.

8. The Grasp Behavior module responds to the update to `me:intent:action` key by starting the moving animation towards the updated location specified on the blackboard in key `me:intent:target`.

9. Diana seamlessly transitions into grasping the blue block.

3.8 Conclusion

The previous section described several interaction scenarios. While the focus was on how each scenario propagated changes to and from the blackboard, and how those changes triggered certain components in the system into action, this section points reader's attention to certain noteworthy aspects of these interactions.

One of the remarkable thing about these interactions is that a lot of it happens in parallel. Where perception is concerned, Diana not only looks for gestures, but is also listening for spoken user commands. Many times, the commands reinforce what the gesture indicated. Sometimes, however, gestures provide information that is hard to communicate by words alone. For example, deixis is best conveyed by pointing at a location and optionally, saying something like "here". The fact that Diana is able to understand both gestures and speech, and combine them into an actionable intent in a realistic way is a distinguishing ability. On the flip side of this scenario, Diana also uses gestures while speaking. This comes in handy when Diana is asking clarifying questions to the user. For example, when Diana says, "Do you want me to grab the red block?", she also reinforces the meaning by reaching for the red block, which eases the conversation by utilizing this redundancy.

Another thing to note is that Diana's interactions run asynchronously. This means Diana can handle interruptions easily, making the communication responsive and adaptable to changes. For example, let's say that Diana is reaching for a block based on a previous command by the user and the user realizes that there is a better plan of action. The user need not wait for Diana's hand to reach the block, and can instead interrupt her. Due to the blackboard architecture and modular design, Diana is able to immediately retract from the previously decided course of action.

Finally, Diana's world knowledge as encoded in VoxML also contributes to her natural language understanding and reasoning about physical objects. An object's habitat conditions its affordances. Therefore, when a block has another block on top of it, the supporting block cannot "afford" to have yet another block on top of it. The rich semantic information provided

by VoxML in this way guides Diana's reasoning about what is a valid course of action and also when Diana suggests alternatives to the user if the user has asked an impossible course of action. Since Diana is situated in her environment, this will create a break in situated grounding, prompting Diana to ask clarification from the user.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

Recent years have seen a huge surge in pervasiveness of virtual assistants like Google Assistant, Alexa, Siri, etc. on our phones and on smart speakers such as Google Home, Amazon Echo, etc. While these agents are definitely smart in that they can answer a variety of queries about the world or its people, they can only do so by listening and speaking. That is, they are limited to a single modality of communication – speech. Speech is an efficient mode of communication, but it can still get very cumbersome or even impossible to communicate everything by speaking. An obvious example is specifying a location. In the utterance "Put the block there", it is very hard to ground the meaning of the word "there" without using a deictic gesture (pointing).

Such virtual assistants reason in isolation from their environment, and as a result, struggle to answer simple questions like "What am I holding?". Most likely, such a question would trigger a default response like "Sorry, I don't understand." In essence, these virtual assistants are not *situated* [13] in their environment. Which is unfortunate since human cognition – the one that these virtual assistants aim to mimic – is believed to be *situated* [12] and in fact, heavily leans on the percepts available to it to render many tasks, including grounding of natural language, easier.

Additionally, these virtual assistants lack an embodiment. Having a body aids the social aspects of an agent, especially if the agent assumes the form of a human. It also opens the agent to non-verbal modes of communication like gestures. Embodied agents have been used for a variety of applications, including military rehearsal exercises [58], distress evaluation [59], etc. as mentioned in Section 2.5.

With the weaknesses of a speech-only assistant in mind, Diana is built to be a multimodal, situated and an embodied conversational agent. Diana is:

multimodal in that she can not only listen to the user but can also *see* the user and understand a variety of hand gestures. She can respond by speaking like most other conversational agents, but she can also gesture to supplement as well as complement natural language utterances. Additionally, she can understand text input as well as output her responses in textual form. She can also detect user's facial expressions as well as respond with those of her own for various emotions like surprise, concentration, joy, etc.

situated in that she employs various perceptual faculties to guide her reasoning. When the user refers to an object and Diana is unable to ground the reference in her environment, this will trigger a clarification dialogue from Diana to resolve the ambiguity. In fact, a major aspect of Diana's reasoning is that it is grounded in visualization (see Section 2.3). To enable this, Diana's reasoning engine – VoxSim – utilizes VoxML, a modeling language to create representations of objects, their attributes, and events enacted over these objects.

embodied as Diana takes the form of a human that is augmented by a variety of animations for various actions like grabbing, waving, idling, etc. She also blinks involuntarily and can follow an object or the user with her gaze. Her understanding of embodiment extends also to the user, as she notices when the user is at a distance as opposed to when the user is close by and takes it as a sign of user engagement.

asynchronous in her interactions which allows her to handle interruptions in a natural manner. For example, a command to grab an object may be interrupted by the user midway by being asked to pick a different object. In this case, the blackboard architecture supporting Diana's inter-component communication ensures that Diana will seamlessly switch over to grab the new object instead without being obligated to complete her current motions.

Creation of an embodied conversational agent such as Diana is a mighty task and involves convergence of a variety of disciplines such as natural language understanding, 3D animation and modeling, dialogue systems, artificial intelligence, computer vision, and machine learning. The contribution of this thesis lies in the detailed description of the entire system in terms of

its architecture and the interplay between its various components. It also lays a foundation for understanding the motivation behind some of the design choices by providing a detailed literature review of the systems that precede it.

4.2 Future Work

The version of Diana described in this thesis operates in blocks world. While this environment is suitable for prototyping, it still leaves a lot to be desired when considering the complexity of real world situations. However, VoxML is amenable to modeling more complicated 3D objects like plates, cups, etc. and hence, Diana can be made to reason about environments having such objects.

Both Diana and the user enjoy a shared space such that Diana can observe the entities in the physical world just as the user can see and point to objects in the virtual world. However, it can be argued that this sharing of space is sometimes awkward, given that both worlds are separated by a TV screen. For example, it is not always obvious to the users that they can point on the screen to point to objects in the virtual world. Such awkward mechanisms can be alleviated by deploying Diana in a Virtual Reality (VR) that would make these interactions much more natural. This, however, comes at the cost of introducing wearable technologies like VR headsets into the system setup. Nonetheless, it is an interesting approach to explore as an alternative.

Another avenue of improvement for Diana could be replacing Google's Automatic Speech Recognition (ASR) with a custom solution tuned to Diana's domain. While the results are decent in the current setup, a finely tuned local system should improve speech recognition accuracy as well as eliminate any latency, thereby ensuring lower response times. A related improvement could be achieved with a better text-to-speech (TTS) engine, especially one that takes prosodic elements into account when creating speech output, thereby providing a human touch to Diana's speech capabilities.

There is also room for improvement by replacing some of the specialized hardware. For example, some work has already gone into replacing Kinect for Windows v2 used in the described

system with Azure Kinect [73]. Azure Kinect improves upon Kinect for Windows v2 by upgrading the RGB camera from HD resolution to 4K UHD resolution, increasing depth camera resolution, providing a 7-mic circular array instead of a 4-mic linear phased array, etc. while being lighter than its predecessor. Using Azure Kinect could increase gesture recognition accuracy while also benefiting system's portability owing to its lighter weight.

Bibliography

[1] Michael A Arbib and Auro Lecci. *The metaphorical brain: An introduction to cybernetics as artificial intelligence and brain theory*. Wiley-Interscience New York, 1972.

[2] M. Rojc and N. Campbell. *Coverbal-synchrony in Human-Machine Interaction*. CRC Press, 2019.

[3] B. Reeves and C. Nass. *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press, 1996.

[4] James Pustejovsky and Nikhil Krishnaswamy. Voxml: A visualization modeling language, 2016.

[5] Nikhil Krishnaswamy and James Pustejovsky. Voxsim: A visual platform for modeling motion language. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 54–58, 2016.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Isaac Wang, Mohtadi Ben Fraj, Pradyumna Narayana, Dhruva Patil, Gururaj Mulay, Rahul Bangar, J Ross Beveridge, Bruce A Draper, and Jaime Ruiz. Eggnog: A continuous, multimodal data set of naturally occurring gestures with ground truth labels. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 414–421. IEEE, 2017.

[8] Isaac Wang, Pradyumna Narayana, Jesse Smith, Bruce Draper, Ross Beveridge, and Jaime Ruiz. Easel: Easy automatic segmentation event labeler. In *23rd International Conference on Intelligent User Interfaces*, pages 595–599, 2018.

[9] R. T. Arn, P. Narayana, T. Emerson, B. A. Draper, M. Kirby, and C. Peterson. Motion segmentation via generalized curvatures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(12):2919–2932, 2019.

[10] Heting Wang. Empathic avatar in task-driven human-computer interaction, an.

[11] Maia Garau, Mel Slater, Simon Bee, and Martina Angela Sasse. The impact of eye gaze on communication using humanoid avatars. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, page 309–316, New York, NY, USA, 2001. Association for Computing Machinery.

[12] C. Lueg and R. Pfeifer. Cognition, situatedness, and situated design. In *Proceedings Second International Conference on Cognitive Technology Humanizing the Information Age*, pages 124–135, 1997.

[13] John Seely Brown, Allan Collins, and Paul Duguid. Situated cognition and the culture of learning. *Educational Researcher*, 18(1):32–42, 1989.

[14] Gale L Martin. The utility of speech input in user-computer interfaces. *International Journal of Man-Machine Studies*, 30(4):355–375, 1989.

[15] Kristiina Jokinen and Michael McTear. Spoken dialogue systems. *Synthesis Lectures on Human Language Technologies*, 2(1):1, 2009.

[16] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[18] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[19] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[22] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[23] J. L. Austin. *How to Do Things with Words*. Oxford University Press, 1962.

[24] Terry Winograd. A language/action perspective on the design of cooperative work. In *Proceedings of the 1986 ACM conference on Computer-supported cooperative work*, pages 203–220, 1986.

[25] David R Traum and Elizabeth A Hinkelman. Conversation acts in task-oriented spoken dialogue. *Computational intelligence*, 8(3):575–599, 1992.

[26] Massimo Poesio and David Traum. Towards an axiomatization of dialogue acts. In *Proceedings of the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues (13th Twente Workshop on Language Technology*. Citeseer, 1998.

[27] Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. Gus, a frame-driven dialog system. *Artificial Intelligence*, 8(2):155 – 173, 1977.

[28] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. A survey on dialogue systems: Recent advances and new frontiers. *SIGKDD Explor. Newsl.*, 19(2):25–35, November 2017.

[29] Wayne Ward and Sunil Issar. Recent improvements in the cmu spoken language understanding system. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1994.

[30] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu, and G. Zweig. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2015.

[31] Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972.

[32] Gerald Sussman and Terry Winograd. Micro-planner reference manual. *AI Memo 203*, 1970.

[33] M. A. K. Halliday. Functional diversity in language as seen from a consideration of modality and mood in english. *Foundations of Language*, 6(3):322–361, 1970.

[34] SHRDLU. https://hci.stanford.edu/winograd/shrdlu/. Accessed: 2020-11-06.

[35] James Pustejovsky. *The Generative Lexicon. A Bradford Book*. Mit Press, 1995.

[36] A. Newell. *Heuristic Programming: Ill-Structured Problems*, page 3–54. MIT Press, Cambridge, MA, USA, 1993.

[37] Iain D Craig. Blackboard systems. *Artificial Intelligence Review*, 2(2):103–118, 1988.

[38] Lee D Erman and Victor R Lesser. A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1975.

[39] H Penny Nii and Edward A Feigenbaum. Rule-based understanding of signals. In *Pattern-directed inference systems*, pages 483–501. Elsevier, 1978.

[40] Barbara Hayes-Roth and Frederick Hayes-Roth. A cognitive model of planning. *Cognitive science*, 3(4):275–310, 1979.

[41] Allan Terry. *The CRYSALIS project: hierarchical control of production systems*. Computer Science Department Stanford University, 1983.

[42] Herbert H. Clark and Edward F. Schaefer. Contributing to discourse. *Cognitive Science*, 13(2):259–294, Apr 1989.

[43] Benjamin Bergen and Jerome Feldman. 16 - embodied concept learning. In Paco Calvo and Antoni Gomila, editors, *Handbook of Cognitive Science*, Perspectives on Cognitive Science, pages 313 – 331. Elsevier, San Diego, 2008.

[44] Friedemann Pulvermüller, Markus Härle, and Friedhelm Hummel. Walking or talking?: Behavioral and neurophysiological correlates of action verb processing. *Brain and Language*, 78(2):143 – 168, 2001.

[45] Olaf Hauk, Ingrid Johnsrude, and Friedemann Pulvermüller. Somatotopic representation of action words in human motor and premotor cortex. *Neuron*, 41(2):301 – 307, 2004.

[46] Teenie Matlock. Fictive motion as cognitive simulation. *Memory & cognition*, 32(8):1389–1400, 2004.

[47] James Pustejovsky and Nikhil Krishnaswamy. Situational grounding within multimodal simulations, 2019.

[48] David Crystal. *The Cambridge Encyclopedia of the English Language*. Cambridge University Press, 3 edition, 2018.

[49] James Pustejovsky, Nikhil Krishnaswamy, and Tuan Do. Object embodiment in a multimodal simulation. In *AAAI Spring Symposium: Interactive Multisensory Object Perception for Embodied Agents*, 2017.

[50] James Pustejovsky. Dynamic event structure and habitat theory. In *Proceedings of the 6th International Conference on Generative Approaches to the Lexicon (GL2013)*, pages 1–10, 2013.

[51] Roshan Singh, Ankur Sonawane, and Rajeev Srivastava. Recent evolution of modern datasets for human activity recognition: a deep survey. *Multimedia Systems*, Oct 2019.

[52] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2:28, 2015.

[53] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[54] Siddharth S. Rautaray and Anupam Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial Intelligence Review*, 43(1):1–54, Jan 2015.

[55] H. Cheng, L. Yang, and Z. Liu. Survey on 3d hand gesture recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(9):1659–1673, Sep. 2016.

[56] M. Asadi-Aghbolaghi, A. Clapés, M. Bellantonio, H. J. Escalante, V. Ponce-López, X. Baró, I. Guyon, S. Kasaei, and S. Escalera. A survey on deep learning based approaches for action and gesture recognition in image sequences. In *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, pages 476–483, May 2017.

[57] Nadia Magnenat-Thalmann and Daniel Thalmann. Virtual humans: thirty years of research, what next? *The Visual Computer*, 21(12):997–1015, Dec 2005.

[58] William R. Swartout, Jonathan Gratch, Randall W. Hill, Jr., Eduard Hovy, Stacy Marsella, Jeff Rickel, and David Traum. Toward virtual humans. *AI Magazine*, 27(2):96, Jun. 2006.

[59] David DeVault, Ron Artstein, Grace Benn, Teresa Dey, Ed Fast, Alesia Gainer, Kallirroi Georgila, Jon Gratch, Arno Hartholt, Margaux Lhommet, Gale Lucas, Stacy Marsella, Fabrizio Morbini, Angela Nazarian, Stefan Scherer, Giota Stratou, Apar Suri, David Traum,

Rachel Wood, Yuyu Xu, Albert Rizzo, and Louis-Philippe Morency. Simsensei kiosk: A virtual human interviewer for healthcare decision support. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1061–1068, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[60] Sotaro Kita. *Pointing: Where language, culture, and cognition meet*. Psychology Press, 2003.

[61] N. Dang. A survey and classification of 3d pointing techniques. In *2007 IEEE International Conference on Research, Innovation and Vision for the Future*, pages 71–80, March 2007.

[62] Ricardo Jota, Miguel A. Nacenta, Joaquim A. Jorge, Sheelagh Carpendale, and Saul Greenberg. A comparison of ray pointing techniques for very large displays. In *Proceedings of Graphics Interface 2010*, GI '10, pages 269–276, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society.

[63] D.A. Bowman, E. Kruijff, J.J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley, 2004.

[64] Carnegie Mellon Motion Capture Database. http://mocap.cs.cmu.edu.

[65] Microsoft. Kinect Unity plugin. http://go.microsoft.com/fwlink/?LinkID=513177.

[66] J. Sell and P. O'Connor. The Xbox One system on a chip and Kinect sensor. *IEEE Micro*, 34 no. 2:44–53, 2014.

[67] Open-source. PyKinect2. https://github.com/Kinect/PyKinect2, 2016.

[68] Google. Google Cloud Speech-to-Text. https://cloud.google.com/speech-to-text/.

[69] Oren Shoham. UnityGoogleStreamingSpeechToText. https://github.com/oshoham/UnityGoogleStreamingSpeechToText.

[70] Affectiva. https://www.affectiva.com.

[71] Affectiva. Emotion AI Overview. https://www.affectiva.com/emotion-ai-overview.

[72] Daniel D Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[73] Azure Kinect. https://azure.microsoft.com/en-us/services/kinect-dk. Accessed: 2020-12-01.