Shape Game
A Multimodal Game Featuring *Incremental* Understanding

Alexander Gruenstein
alexgru@csail.mit.edu

1 Introduction

The goal of much research in speech understanding systems is to design interfaces which allow for humans to interact with computers in a more natural and intuitive fashion. Intuitively, speech is a very natural, convenient, and fast way of communicating information – and people produce and understand speech with little overt effort. Despite the ease with which humans communicate with *one another* using speech, many find it frustrating to communicate with *computers* via speech. This frustration can be linked to many limitation of speech understanding systems, including limitations in the accuracy of automatic speech recognition, parsing, referent resolution, and a general lack of world knowledge. But even if these challenges were overcome, the vast majority of speech understanding systems would likely still feel quite unnatural, owing to the strong *turn-based* model that most are designed around. Typically, speech understanding systems are designed following a model in which users are expected to formulate complete, well-formed ideas (typically in the form of commands or queries) in a fully fluent utterance. The reality is, most people don't talk this way – those of us who can clearly stand out: television personalities like American Idol host Ryan Seacrest or news anchors like Tom Brokaw come to mind.

The reality is, most of us aren't very good at staring at a television camera – or a computer monitor – and speaking fluently, without immediate feedback that anyone (or any*thing*) is listening or understanding. In my experience designing and testing several different multimodal speech understanding systems, this strict idea of turn taking has always stood out as being extremely awkward whenever naive users try to interact with a system. One of the most challenging aspects of designing speech understanding systems is giving a sense of what the system can understand, especially conveying to a user why the system has made an error in understanding. Humans are quite good at conveying what they have or have not understood about what another person has said, and one strong mechanism for signaling this is through immediate feedback to a person as they are speaking. This feedback comes in many different forms, including gestures, facial expressions, and backchanelling; and it can play a vital role in helping a speaker know when to rephrase, speak more slowly, and so on. However, the strict turn-taking structure of most speech understanding systems makes this sort of subtle, yet critical, interaction impossible. After attempting to formulate a fluent utterance, users are left to then try to figure out what went wrong and why.

In this report, I describe *Shape Game*, a novel speech understanding system which provides non-intrusive incremental feedback to users as they speak in the course of the game. While extremely simple, *Shape Game* affords both a test-bed and framework for the types of capabil-

ities which could be integrated into more complex systems. The system is multimodal, in that it accepts speech input which modifies a game board which is shown to the user using a computer screen. Feedback to the user is provided graphically, as they speak, through the use of highlighting, shading, and flashing. Designing even such a relatively simple game with just a few feedback strategies proved surprisingly challenging, and I believe the work yields some interesting insight which can be incorporated into more complex projects.

In this report I'll cover some related work which inspired *Shape Game*. I'll then describe the various components which make *Shape Game* possible, and provide an analysis of a small user study. Finally, I wrap up, and discuss future directions of exploration.

2 Related Work

James Allen's group at the University of Rochester has recently published papers on incremental speech understanding, focusing on a domain called the "fruit carts" domain [see, for example, (Allen et al., 2005; Aist et al., 2006)]. Figure 1 shows the on-screen display used in this domain. Subjects are given a card similar to the top half of this figure, in which several different locations contain shapes filled with fruits of various colors, oriented at various rotations. Subjects then speak commands into a microphone, until their (initially unpopulated) map looks like the card they were given. While the group's goal is to eventually build a speech understanding system which can perform this task, I am unaware of the existence of such a functioning system. At the moment, all data is collected via *wizard-of-oz* methods, where an unseen human manipulates what the subjects see on the screen in response to their commands.

This domain is particularly interesting because it is ripe for providing incremental feedback to users as they speak. The authors have observed the users tend to give small, incremental commands; and that they use system feedback to judge what future steps should be taken. The following actions are possible in the domain, with examples of how they are expressed [quoted directly from (Allen et al., 2005)]. And figure 2 reproduces an example of speech data collected through the experiments.

```
1 okay so
2 we're going to put a large triangle with nothing into morningside
3 we're going to make it blue
4 and rotate it to the left forty five degrees
5 take one tomato and put it in the center of that triangle
6 take two avocados and put it in the bottom of that triangle
7 and move that entire set a little bit to the left and down
8 mmkay
9 now take a small square with a heart on the corner
10 put it onto the flag area in central park
11 rotate it a little more than forty five degrees to the left
12 now make it brown
13 and put a tomato in the center of it
14 yeah that's good
15 and we'll take a square with a diamond on the corner
16 small
17 put it in oceanview terrace
18 rotate it to the right forty five degrees
19 make it orange
20 take two grapefruit and put them inside that square
21 now take a triangle with the star in the center
22 small
23 put it in oceanview just to the left of oceanview terrace
24 and rotate it left ninety degrees
25 okay
26 and put two cucumbers in that triangle
27 and make the color of the triangle purple
```

Figure 2: Example of a user interacting with the Fruit Carts domain, reproduced from (Aist et al., 2006)

1. Select an object ("take the large plain square")
2. Move it ("move it to central park")
3. Rotate it ("and then turn it left a bit that's good")
4. Paint it ("and that one needs to be purple")
5. Fill it ("and there's a grapefruit inside it")

3 Shape Game: Functional Description and Design

In designing *Shape Game*, I used the Fruit Carts domain as inspiration, however I developed a greatly simplified version of the "game" which could realistically be implemented given the

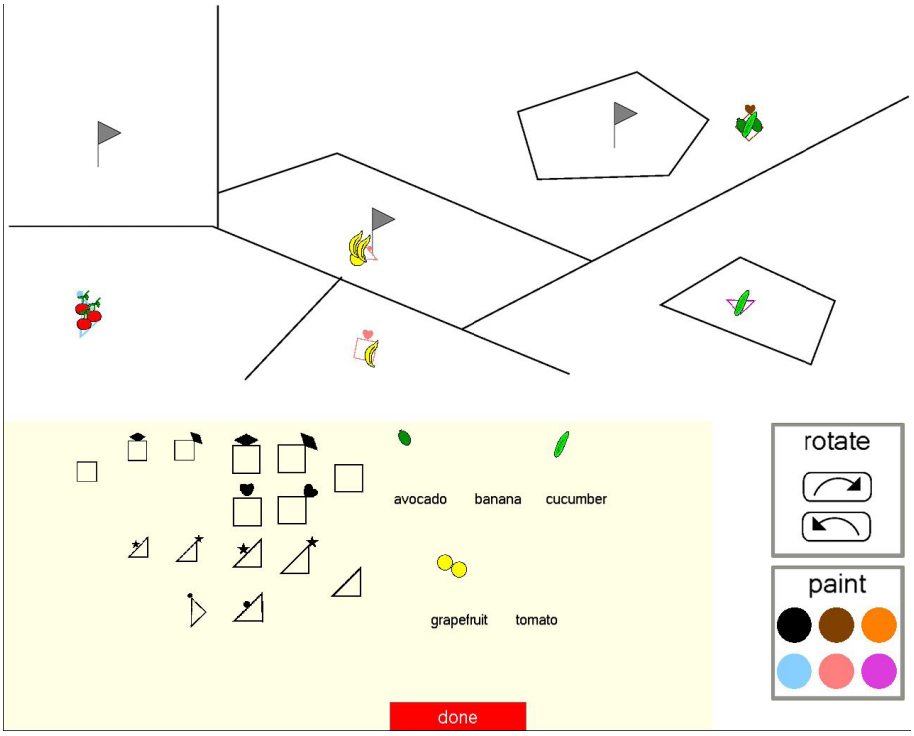

Figure 1: Fruit Carts domain screenshot, reproduced from (Allen et al., 2005)

time constraints. By far the best way to understand the game is by watching a video of the interaction, and actually playing the game. Please see http://people.csail.mit.edu/alexgru/shapegame/.

Figures 3, 4, and 5 show examples of the game being played. The game supports three different commands:

1. *Selection*, as in "select the small green circle"
2. *Put/Drop* as in "put it in slot five"
3. *Make* as in "make it blue"

The game is very easy to play. A player uses these three commands to first select one of the 12 shapes along the bottom (called *morphable* shapes); she must provide enough attributes to uniquely identify a shape. Once a shape is selected, it is dropped into one of the numbered slots. Then, any number of "make" commands are used to change various attributes of the shape. The goal is to make all six shapes in the numbered slots look like the six shape *templates* above them. Currently there is no scoring, but it would be very natural to set a time limit, or to count the number of commands used to accomplish the task.

3.1 Incremental Understanding

Shape Game is designed to process incremental recognition results from the speech recognizer, as the user speaks. It tries to indicate that it is understanding (or not understanding) what the user is saying as quickly as possible, giving the user the chance to correct the system's understanding. Understanding is indicated through visual feedback to the user as she speaks commands, mostly through the use of highlighting. While designing the incremental processing component, it became apparent that there were two stages to the incremental processing. First, the system should indicate that it has *understood* what has been said so far, while still giving the user a chance to *revise* that understanding. Only once the user has indicated (usu-

ally implicitly) that the system's understanding is correct, should the system actually *execute* a command.

Each of figures 3, 4, and 5 demonstrate different techniques *Shape Game* uses to provide incremental visual feedback. Figure 3 shows the most straightforward use of visual feedback: the incremental processing of the following selection command: *select the red rectangle [pause] the large red rectangle*. As this command is given, highlighting is used to show the set of shapes which match the current description. While speaking, the user immediately notices that "the red rectangle" in fact describes two rectangles, and is able to revise the selection simply by revising the description of the shape. Notably, in a typical speech understanding system, the user would have to hold down the hold-to-talk button, say the command, and then let go before seeing the result. Then she would have to press and hold the button again to make the revision. In *Shape Game*, she can just hold down the button the whole time she's speaking, and repair her utterance on the fly.

Figure 4 shows how highlighting is used during a sequence of commands: *drop it in slot five slot six and make it small*. As soon as the user says "drop" all of the possible drop slots are highlighted, indicating the system's expectation that we are in the process of a drop command. Then, once "slot five" is mentioned, this slot is highlighted. However, the drop command is not actually executed at this point – which gives the user a chance to revise. The user changes his mind, noticing that the slot to the right is actually a better match, and makes a correction mid-stream simply by saying "slot six." This slot is then highlighted. Once the user then begins the "make" command, the actual drop occurs; as moving on to a new command is taken as an implicit signal that the user is happy with the system's understanding of the previous command. Next, the utterance of *make it small* is given and the triangle is made smaller; however it is rendered translucently, to indicate that this is still a tentative change. At this point, the user could still say hesitate and say, for example, "blue" to indicate that they actually want to change the color instead of the shape. Finally, the user releases the hold-to-talk button, signalling that this command is complete, and the triangle becomes opaque, to indicate that the command has been executed.

Finally, figure 5 shows the last type of incremental feedback which can be provided in *Shape Game*. Here, the user first issues this command: *select the large triangle*. However, this does not uniquely identify a shape, as there are two large triangles. Oblivious, the user begins the next command: *drop it in ....* However, as soon as the system understands "drop", it indicates visually that it doesn't know which shape to drop by flashing question marks in the locations of the two large triangles. Thus, whereas in a typical speech understanding system the user would have to complete the "drop" command before receiving feedback that it can't be executed, with incremental understanding the previous utterance can be immediately corrected. This feels extremely natural: a human playing the role of the computer would likely try to interrupt the user to indicate the problem before moving on.

4 System Architecture

Now that I've introduced *Shape Game* on a functional level, I'll describe the architecture underlying it. The overall architecture can be found in figure 6. The interface is based on a client-server architecture, in which the speech and natural language processing components reside on a server. The interface to the game itself is web-based, and can be accessed in a standard web browser. A Java applet streams audio to the speech recognizer, while AJAX (Asynchronous Javascript And XML) techniques are used to dynamically update the game interface shown on the web page. The large gren button at the top of the page is a hold-to-talk button,

Figure 3: Screenshots of the system as it incrementally responds to a "selection" command: *(1) select (2) the red (3) rectangle (4) [pause] the large (5) red rectangle.* (Numbers indicate points in the incremental recognition in which each screenshot occurs.) The user gets immediate feedback about which shapes the given constraints select.

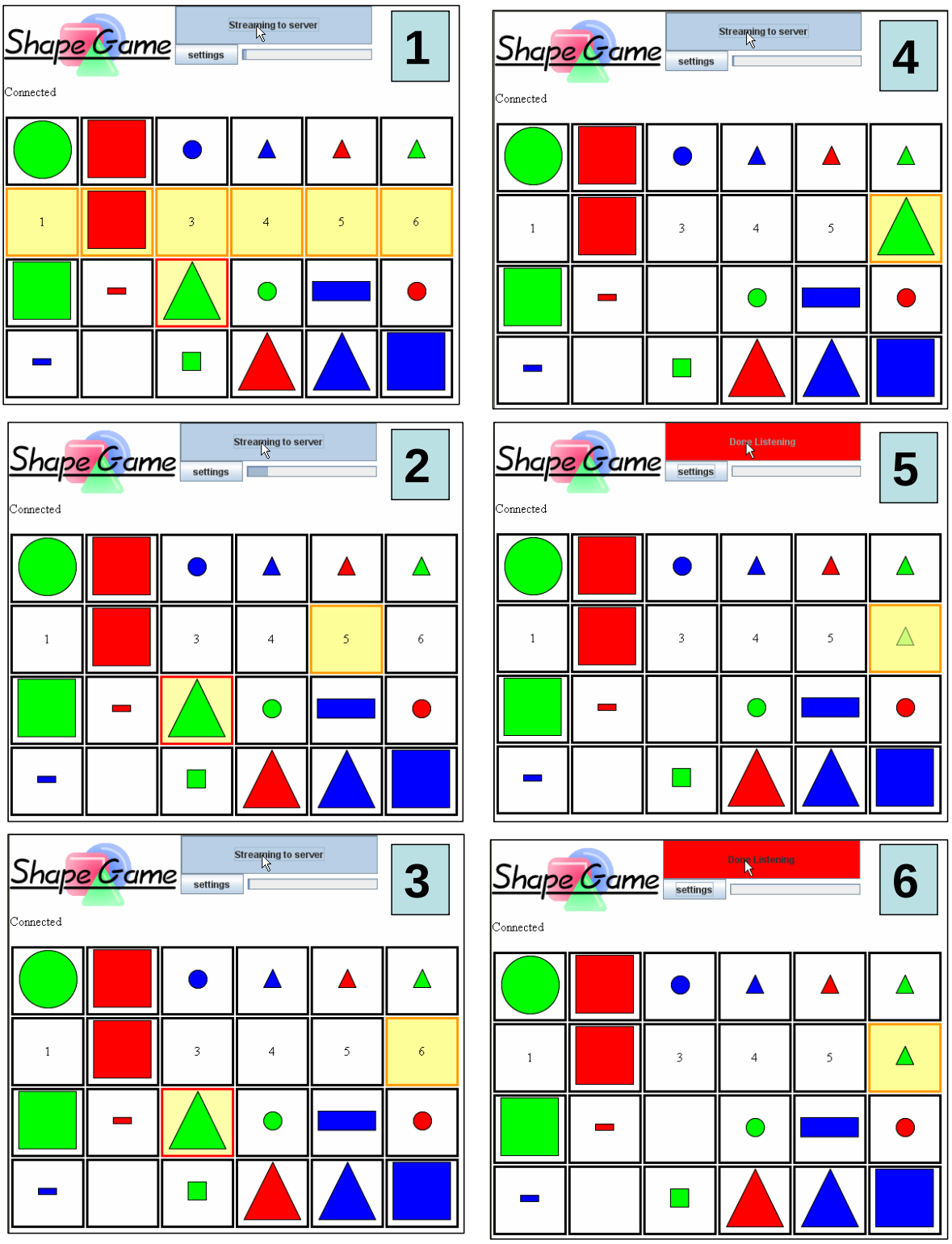

Figure 4: Screenshots of the system as it incrementally responds to a "drop" command followed by a "make" command: *drop (1) it in slot five (2) slot six (3) and make (4) it small (5) [release hold-to-talk-button] (6).* We note that the selected slot for the drop is highlighted, but the user can still change her mind mid-utterance to change it; it is not until the beginning of the "make" command that the actual drop action occurs. Similarly, during the "make" command, a small triangle is shown in a translucent green color until the button hold-to-talk button is released, at which time it becomes solidly colored.

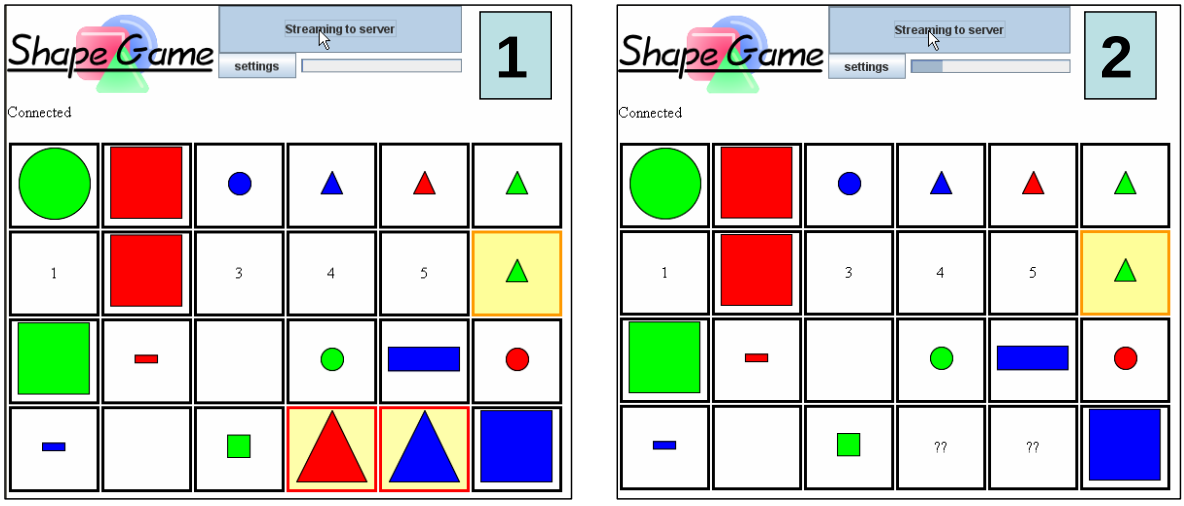

Figure 5: Screenshots of the system as it incrementally responds to the following utterance: *select the large triangle (1) drop (2) it in*. The selection is ambiguous (it selects two shapes), as soon as the user begins the "drop" command, the system flashes question marks over those two shapes to indicate that it doesn't know which one to drop. This allows the user to make a correction immediately, without having to complete her command.

which the user must hold down while speaking. The architecture was developed for the *City Browser* application (Gruenstein et al., 2006), and now serves as a platform for several multimodal dialogue systems in our lab.

On the server, we use the SUMMIT automatic speech recognizer (Glass et al., 1999), which interacts with our Java Servlet based Game Engine via the GALAXY framework (Seneff et al., 1998). Incremental speech results are sent as speech is processed from the recognizer to the Game Engine, where processing occurs. Details of various key aspects of the system follow.

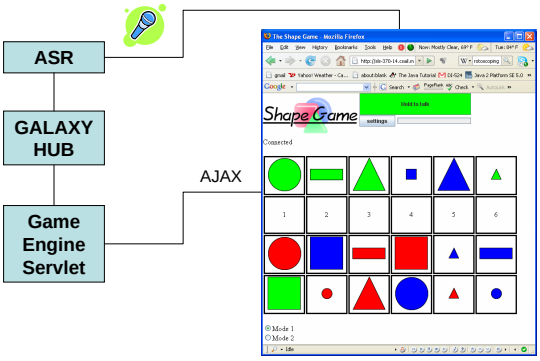

Figure 6: Shape Game Architecture

4.1 Incremental Recognition

In order to obtain incremental recognition results, I had to make modifications to the SUMMIT recognizer, which is implemented in C++. Typically, in the recognizer, speech is processed as it is received, however the best (or n-best) recognition hypotheses are not determined until the utterance is complete. In order to calculate the best overall candidate hypothesis for the utterance, it is necessary to have the entire utterance, as the language model will be used as a constraint. However, as we are processing the utterance, it is possible to determiine our "best guess" so far. We employ a method which is almost identical to the viterbi backtrace procedure used once the utterance is complete. However, while the normal backtrace enforces the constraint that the final state be a *final* state (one that we have evidence showing it should occur

at the end of an utterance), this constraint is relaxed when we perform the search while still processing the utterance.

The next important question is how often to do the calculations required to find the best partial hypothesis. In a very small domain like Shape Game, I've found it to be no problem to update the partial hypothesis as soon as any new chunk of data is received for processing. This means that the calculation occurs quite frequently – sometimes on the order of every few milliseconds. However, new partial results are only sent to the Game Engine if the new partial hypothesis differs from the previous one. This means that in Shape Game, partial results are usually send after every word, or almost every word. In a larger domain, this might represent a significant overhead, and a balance would have to be found between the need for partial results and computation.

4.2 Language Model

I used a very small context free grammar in the Java Speech Grammar Format (JSGF) as the speech recognizer's language model. Conveniently, it is also possible to embed semantic tags in the grammar which are output alongside the recognized words. This means that I did not have to use a separate parser to understand utterances, instead I am simply able to discard the actual recognized words and look only at the semantic tags which are output in the stream. The entire grammar is given in figure 7.

The grammar is also notable in that it allows many different types of *false starts* and *corrections*. At the same time, it allows an arbitrary number of commands to be looped together. The following utterance illustrates the flexibility of the grammar:
select the large green select the large green blue red triangle red circle and drop it in slot five six slot four and make it green blue a square make it blue select the small large red rectangle
This flexibility allows for a large amount of freedom on the part of the user. The down side

```
#JSGF V1.0;
grammar ShapeGame;

public <top> = (<command> [and])+ ;

<command>       = <select>
        | <drop>
        | <make>  ;

<select>        = (select | choose) {[command=select]} <shape_desc>+ ;

<drop>          = (drop | put) {[command=drop]} it in <slot>+ ;

<slot>          = slot <number>+ ;

<make>          = make {[command=make]} it [a] <shape_attr>+ ;

<shape_desc>    = [the] <shape_attr>+ ;

<shape_attr>    = <size> | <color> | <shape_type> ;

<size>          = <size_adj> [sized] ;

<size_adj>      = (small | tiny) {[size=small]}
                | medium         {[size=medium]}
                | (large | big)  {[size=large]} ;

<color>         = red   {[color=red]}
                | green {[color=green]}
                | blue  {[color=blue]} ;

<shape_type>    = (rectangle | bar) {[shape_type=rectangle]}
                | square            {[shape_type=square]}
                | circle            {[shape_type=circle]}
                | triangle          {[shape_type=triangle]}
                | one
                | shape ;

<number>        = one {[number=1]}
                | two {[number=2]}
                | three {[number=3]}
                | four  {[number=4]}
                | five  {[number=5]}
                | six   {[number=6]}
                ;
```

Figure 7: The context free grammar used for the recognizer's language model. Words enclosed in $< >$ are nonterminals. Text enclosed inside curly braces $\{\}$ are semantic tags.

of it is that the language model is not very well constrained. In this very narrow domain, this does not cause problems. However it would likely lead to a degradation of accuracy in a domain with a larger vocabulary and more complex rules. Recognition difficulties could be mitigated by using a *probabilistic* (or weighted) context free grammar, trained with collected user data (or via intuition). This could be used to lend greater weights to false starts and corrections where they are observed to be more frequent. In addition, I believe that adding terminals like "uh", "um" and "i mean" which typically signail repairs, as well as an overt modeling of pauses, could improve performance. So, too, could the utilization of prosodic information. Alas, this speculation all represents future areas of exploration.

4.3 Game Engine

I have established the mechanism by which the recognizer outputs partial results, and shown how our language model can be used to directly provide "semantic" information. I now turn to the Game Engine, which processes these partial recognition results, embedded with semantic tags, and updates the game display. The Game Engine is implemented as a Java servlet, which updates the game displayed in the browser by sending XML messages to the client browser. The browser uses javascript to dynamically render the web page, based on the contents of the messages it receives from the servlet.

The Game Engine must process incremental recognition results sent from the recognizer as they occur. Two typical sequences of partial results from the recognizer are shown in figure 8. The game engine processes these partials by throwing away the recognized words and processing the list of semantic tags. The list of semantic tags is segmented into separate lists for each command in the sequence. The index of the currently active command is stored after each partial result is processed. This is critical, as a partial result might contain several commands (if the user holds down the hold-to-talk button for a long time), however many of these commands may have already been *executed*.

To determiine the the specifics of each command, a simple method is employed. The Engine simply iterates over each semantic tag in that command's list, in the order the tags were recognized, and sets its value in a hashtable. That way, the *most recent* value for that semantic tag will be stored in the hashtable after we finish iterating over all the semantic tags in the list. This simple method accounts for most false starts and corrections, simply by throwing away earlier information whenever it is replaced.

While a command is active, the system indicates that it *understands* the command, however it doesn't execute it. This is done multimodally, as was described extensively in section 3. When the user indicates that she is satisified with the command, either by moving on to a new command or by letting go of the hold-to-talk button, the system *executes* the command and advances the stored index of the currently active command. This is important both because a user may revise a command, and because the current partial recognition result is often error prone towards the end of the hypothesis. Errors arise because the recognizer is quite aggressive in matching the best word in its vocabulary with a what it hears; so, often as you are in the middle of saying a word, the recognizer will already be trying to make its best guess about what that word is. As the partial results in figure 8 show, this often leads to very brief errors in the partial results – we note that "slot three" is initially hypothesized to be "slot six," but this is subsequently revised. While it might make sense to ignore the last word or two in the hypothesis all together, in our small domain the recognizer is often, in fact, correct – and this means that sometimes the system feels as though its almost reading your mind, understanding words you have just begun to say. Since we can update the display pretty rapidly, the user may not even notice if the wrong slot is highlighted for

```
select [command=select]
select [command=select] small [size=small]
select [command=select] the
select [command=select] the big [size=large]
select [command=select] the big [size=large] red [color=red]
select [command=select] the big [size=large] red [color=red] rectangle [shape_type=rectangle]

drop [command=drop] it
drop [command=drop] it in slot
drop [command=drop] it in slot six [number=6]
drop [command=drop] it in slot three [number=3]
drop [command=drop] it in slot three [number=3] slot
drop [command=drop] it in slot three [number=3] slot four [number=4]
```

Figure 8: Partial recognition results for two commands: *select the big red rectangle* and *drop it in slot three slot four*

a fraction of a second.

The Game Engine minimizes traffic over the HTTP connection to the web browser, as this link is potentially slow. In order to do this, it stores the state of the display locally and only sends messages to the client when the display must be changed. In addition, it knows the position of each shape, and generates a random set of shapes at the beginning of each game. Shapes are generated so that no two shapes are the same anywhere on the board. This guarantees that each *morphable* shape on the bottom can be uniquely identified, and that always at least one move is required to change a morphable shape to match a *template*.

4.4 Game Display

The game is displayed as standard HTML in a web browser. Javascript is used to update this diplay as new messages are pushed from the server. The shapes for the game were created using a script which produced SVG (Scalable Vector Graphics) descriptions of each shape which were then converted to PNG files suitable for display in a web browser. Clearly, the responsiveness of the system is highly dependent on the speed and latency of the user's network connection – and given that we are interested in incremental understanding, latency is a huge potential pitfall. I have tested the system in three scenarios: both server and client on the same computer, client on MIT's wireless network, and client on my cable modem at home. I've found the three different locations to be indistinguishable with regards to responsiveness.

5 User Evaluation

I performed a small user evaluation of the *Shape Game* by asking naive users to interact with it. 1 pilot user tried the system early on, and helped me to identify the need to be more careful to only show understanding until a command is ready to be executed. Early versions of the game immediately executed commands as they came in. While this model works fine for shape selection, it is confusing and error prone for dropping shapes and changing their attributes

After refining the system based on the pilot user's experiences, I asked 4 subjects to use *Shape Game*. Each user was given a brief demonstration of how to use the system, and then asked to play the game in two different modes marked as "Mode 1" and "Mode 2" on the web page. In Mode 1, incremental understanding was performed as has been descriped in this report. In Mode 2, no incremental feedback was given to users; instead, they could say as many commands as they liked in a row while pressing the hold-to-talk button, but would not see the results of these commands until releasing the button. Three subjects played the game first in Mode 1 and then Mode 2, while one sub-

ject played in the reverse order.

After using the system, I asked each subject to give me feedback about whether they liked the game, and about which mode they preferred. Three indicated that they prefered the mode with incremental feedback, while one thought they were both different but equal. This user perceived the incremental understanding mode as more of a "beginners" mode, while the mode without incremental understanding was thought to be more of an "advanced" mode. I think this observation came out of the fact that users tend to speak a little more slowly in the incremental mode, as they wait for feedback from the system before moving on. In the other mode, they felt as though they were going faster because they weren't afraid to just say several commands and hope for the best. While accuracy in this domain is high enough that this is a viable strategy, in a larger, more errorful domain, users would likely be afraid to partake in such behavior.

In addition, I received feedback that the incremental mode would be more helpful if the task was more difficult. Along these same lines, I also received feedback that this might be a good game to play in a non-native language. It would be a good way to practice vocabulary, and at the same time the incremental mode might be more useful because the task would be more difficult for a non-native speaker. Unfortunately, all of my subjects were native english speakers.

Finally, I observed that in both modes, people did make use of the system's ability to handle disfluences, false starts, and corrections. Subjects tended to have more successful interactions because of these capabilities; incremental mode is a nice way for them to gain confidence that this sort of processing is possible.

6 Conclusion and Future Work

I have presented *Shape Game*, a novel multimodal speech understanding game, which features incremental understanding. I have discussed the various components which make this application possible. I have shown that users do tend to prefer the incremental mode, even in this simple task. I believe that in more complex tasks, incremental understanding with graphical feedback has the potential to play a more important role.

There are several paths to go down in furthering work on this project. After getting a very positive response from people in SLS who are interested in games to help people learn new languages, a very interesting path to go down would be to develop the game further in several different languages. The game could be used to help build vocabulary, and need not be limited to shapes. Different sets of vocabulary could be used (animals, foods, cars, etc) in this game, or a modified version of the game. In addition, a stage where vocabulary is incrementally introduced could be useful. In any case, the work I've presented here serves as a nice model for a fairly painless way to create new kinds of speech understanding games. It would be very interesting to see how well the partial recognition results would scale up to more complicated games, and to determine if they are useful in language acquisition. It is especially difficult to formulate a complete, fluent sentence in a foreign language, so the ability to make repairs as you speak and have the system show that it understands, could be extremely useful.

I am also interested in taking what I've learned here and incorporating it into multimodal dialogue systems which are an order of magnitude more complex than *Shape Game*. The same principles of showing understanding without executing commands immediately could be extremely useful in more complex domains. Scaling it up could be quite complicated, especially as the partials would likely become much more error prone. It would be key to find ways to give feedback about what has been understood so far, without necessarily indicating that you've understood the most recent few

words.

7 Acknowledgements

Stephanie Seneff, Chao Wang, Lee Hetherington, and Scott Cyphers all provided valuable help in conceiving and implementing various aspects of the game. The volunteers who played the game also provided valuable feedback: Anna Liess, Sean Liu, Ian McGraw, Asthma Mohammad, and Justin Weinstein-Tull. Finally, thanks to Ali Mohammad who designed *Shape Game*'s logo, after seeing my own pathetic attempt.

References

Gregory Aist, James Allen, Ellen Campana, Lucian Galescu, Carlos A. Gómez Gallo, Scott C. Stoness, Mary Swift, and Michael Tanenhaus. 2006. Software architectures for incremental understanding of human speech. In *Proc. of INTERSPEECH-ICSLP*.

J. Allen, G. Ferguson, M. Swift, A. Stent, S. Stoness, L. Galescu, N. Chambers, E. Campana, and G. Aist. 2005. Two diverse systems built using generic components for spoken dialogue (recent progress on TRIPS). In *Proc. of the ACL Interactive Poster and Demonstration Sessions*.

Jim Glass, T. J. Hazen, and I. Lee Hetherington. 1999. Real-time telephone-based speech recognition in the JUPITER domain. In *Proceedings of ICASSP*, March.

Alexander Gruenstein, Stephanie Seneff, and Chao Wang. 2006. Scalable and portable web-based multimodal dialogue interaction with geographical databases. In *Proc. of INTERSPEECH*.

S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. 1998. Galaxy-II: A reference architecture for conversational system development. In *Proc. ICSLP*.