Simon Says, Lego-man Do...
AR Characters Educating Novices in Programming

Adrian Clark
ajc191@cosc.canterbury.ac.nz

Mindy Marshall
mjm198@cosc.canterbury.ac.nz

Karthik Nilakant
kni19@cosc.canterbury.ac.nz

Lara Rennie
lre15@cosc.canterbury.ac.nz

Department of Computer Science, University of Canterbury, Christchurch, New Zealand

29th September 2004

Abstract

This paper presents a system which exploits the advantages of Augmented Reality (AR) combined with speech recognition to assist in teaching computer novices or children about central computer programming concepts. 3D animated characters are employed to achieve this, and the user can instruct these in a series of primitive instructions, carry out other simple pseudo-programming tasks, and define methods containing a particular set of commands. Although final implementation is incomplete, preliminary results indicate that the application shows promise. Further development of the system and some formal empirical trials would be valuable to confirm this hypothesis, which, if correct, could see a similar systems being used eventually in educational contexts.

1 Introduction

The suitability of using computer technology for educational purposes has long been recognised. In an effort to extend this into the realms of Augmented Reality (AR), we developed a system to teach computer novices and children about computer programming concepts by using AR 3D characters. The ARToolkit system was used to achieve this, as it enables the difficult parts of visual tracking to be performed by a standard library so focus can be placed on the AR application, rather than with the implementation of the low-level necessities of AR. With the novice target user in mind, it was decided that enabling speech recognition would allow the most natural mode of interaction with these characters. Speech synthesis would also be provided. The user would learn programming concepts by instructing the characters to perform simple tasks. Functionality would be provided such that users could teach the characters to perform a particular task composed of a series of primitive operations or sub-tasks.

1.1 Project Motivation

The relative infancy of the AR field has meant that little research or exploration has been undertaken in analysing the benefits of harnessing AR for education. The effectiveness of AR has been investigated in many other domains however, such as in medical applications and in annotating environments like museums or work environments for mechanics with information(Azuma & Bishop 1994). This lack of work on the educative potential of AR was a central motivating factor behind the project, especially as it is hypothesised that the strengths of AR would see AR applications lend themselves well to education.

One of these strengths of AR is the ability of AR to provide a collaborative environment. This is particularly relevant when teaching computer programming as it is typically difficult to achieve when using a computer. Users are restricted to looking at either the screen or each other, but not at both at the same time. The addition of speech recognition also helps here as it avoids the problem of typical computer use that only one person can control the input at a time.

Also important when designing an educational tool is its general appeal, especially when the target audience is children. AR is an appealing technology, and will become even more so as the technology

develops and head-mounted displays become more comfortable and accepted.

It is thought the natural interaction facilitated by AR and speech recognition would also be beneficial to the particular audience of this program. Rather than having to learn complicated commands, or abstract syntax requirements, the user would be able to use natural language to achieve results.

Finally, AR takes advantage of innate human spatial ability. If this system was to be written for a normal computer screen, it would be much more difficult and less intuitive for the user to manoeuvre the character around, and it would be easy to 'lose' the character off the screen. The 3D ability of AR would make this no longer a problem.

Beyond the desire to explore the capabilities of AR with regards to education, the choice of the specific educational domain of computer programming was motivated by the difficulties many people have in grasping central programming concepts. As identified by Ramalingam, LaBelle & Wiedenbeck (2004), computer science is a discipline with a high dropout and failure rate. Any attempt to alleviate this would hence be welcomed by computer science educators.

The following section outlines the related research already performed in the areas of education and speech recognition with AR, as well as on animated characters in AR. Based on this, the goals for our project are detailed in section 3. The implementation is then detailed, with the breakdown of the system into three linked components. Finally, a discussion of the particularly challenging aspects of this project as well as the results of some exploratory user evaluation are offered, before this report concludes with ideas for future work in this area.

2 Related Research

This project brings together three different realms of computer science, namely speech recognition, AR animation including characters, and computer programming education. Related previous studies have hence been researched and are summarised by topic below.

2.1 Learning Computer Programming

The high rate of drop-outs in computer programming courses has motivated much research on how to teach novices central computer programming concepts. Mayer's (1981) paper on the psychology of this begins by discussing the process with which the mental model of computer concepts is built. The learner must pay attention to some stimulus so that it can enter their short-term memory, then try to attach it to some pre-existing ideas so that it can enter their long-term memory. Without any related pre-existing concepts, the learner must rote-learn each new informational chunk separately, something which is more difficult, and of less use as it makes it harder for the learner to transfer such knowledge to new situations. Two techniques are discussed to help learners relate new computer programming concepts to existing knowledge so that their assimilation of new ideas is much easier and swifter.

One of the techniques offered to help with this step is a concretisation of the task at hand. In the realm of mathematics, success was demonstrated with this by teaching students subtraction with the aid of wooden sticks. This is typically hard to achieve with teaching computer science, as programming usually involves dealing with unfamiliar and intangible objects. A suggestion offered by Mayer (1981) is representing output as a message note pad, and input as a ticket window with data lining up waiting to be processed. Studies showed the success of these concrete examples in any activity requiring some transfer of the understanding of these computer concepts to slightly different scenarios. It is hoped that, in our project, the actions of the physical character would illustrate concepts, like iteration and encapsulating a group of instructions as a method, in concrete ways, and hence help the learner to better understand them.

The other technique discussed was that of ensuring the user had to explain the information in their own words. This is shown in the paper to have been effective in helping people assimilate new information, as by they are forced to verbalise the information in words already linked to previous experience and ideas, thus assisting to link the new concept to these. In our project, the principle input method will be speech recognition. This allows the person to verbalise what they want the character to do, and will be especially valuable in that it will try and use natural English as much as possible. In this way the concepts will be linked to already known commands. Our system will therefore hopefully be able to achieve its stated goal as it utilises both these techniques.

A second body of research in the area of the psychology of learning to program was carried out by Ramalingam et al. (2004). This investigated self-efficacy and mental models when learning to program. Self-efficacy is described as people's abilities to judge their ability to perform correct actions,

and is very important to successful performance. Mental models are representations of real world systems, and an accurate well-developed model is supposed to increase self-efficacy. In Ramalingam et al.'s (2004) study, an experiment was carried out to analyse the self-efficacy and mental model skills of a group of students before commencing and after completing a computer science course. This found that self-efficacy had an significant increase over the semester the course was taken, however their study was unusual in that there was found to be no significant relationship between self-efficacy and mental models. Our project is designed to give the students a high level of self-efficacy when beginning to work with the system, as the interface is designed to be non-threatening and fun. The ease in which the character can be manipulated is also designed to increase confidence. The use of speech as an input device means there is no need for students to learn a new or unfamiliar mode of input, as small children may not be proficient typers, so this will further increase their self-efficacy.

Children and novice computer users are not the only people that it is hypothesised our system will assist. To this end, research was undertaken into previous studies on teaching computer programming to students with a learning difficulty such as dyslexia. In Powell, Moore, Gray, Finlay & Reaney's (2003) paper, the issues and possible solutions when teaching computer programming to dyslexics (1.2-1.5% of tertiary students) are discussed. The main problem faced by dyslexics is their ability to generate and correct computer code. For dyslexics who want to learn computer programming, it can be seen that our system could help them to learn some of the programming concepts without suffering the extra cognitive load of trying to read and write computer code, something especially trying for them. In this way, our system could be seen to have benefits to people with disabilities that makes it difficult for them to deal with a conventional interface.

2.2 Teaching and Multimedia

One method by which learning can be improved is by presenting material in new ways, utilising the latest technologies available. H (2002) presented a introductory computer science course designed to teach non-computer scientists programming by using animations and building simple 2D and 3D virtual worlds. The course starts with simple animations and introducing concepts and builds up to very simple programming by its conclusion. The aim in creating this university course was to make programming fun by presenting it through objects in a visual and animated way. The course contained a short section teaching HTML, then moved onto sorting, animations using JAWAA, StarLogo, how to use Alice to create a 3D world, then finally Karel++ for some gentle programming. The authors found that overall the course was a success although the JAWAA unit was not as popular and some changes to the course were planned as a result of the evaluation. This research supports our hypothesis that presenting an interesting, enjoyable 3D teaching system will help make learning to program fun.

The suitability of Virtual Reality (VR) for education was investigated by Bricken (1991) who found it naturally suited this application. This is firstly because of its experiential nature, with the learner perceptually involved, and experiencing something that they may not be able to otherwise. This also applies, albeit to a lesser extent, to AR. The paper points out the heavy consensus by educational theorists on the importance of such experiential learning. It is hoped that the 'real' 3D character of our project will permit the learner to experience for themselves the effect individual simple programming commands can have.

The intuitive nature of VR, and hence AR, is also discussed in this paper, as users can move, talk, gesture and manipulate intuitively. Another attribute of VR that was mentioned as an explanation for its suitability to the education domain is its ability to create a shared environment. Collaborative learning is enjoyed by students, and is often more effective as learners can discuss their ideas and build on each other's knowledge. AR, however, allows the users to interact with each other and the object under scrutiny at the same time, as visibility is not obscured. It is hoped that our system will facilitate such a collaborative approach.

Construct3D is an actual implementation of an AR system for education. H (2002) created the 3D geometry construction tool to take advantage of the spatial abilities of a 3-dimensional interface to teach geometry. However, they are yet to evaluate their key hypothesis that "Actually seeing this in 3D and interacting with them can understand and enhance a student's understanding of three-dimensional geometry."

2.3 Speech Interaction

Speech recognition (SR) technology has been in development since the 1980s. However, Deng & Huang (2004) identify a number of reasons to explain why this technology has not emerged as a

mainstream form of interaction. Their paper states that the main requirement for the widespread acceptance of speech recognition is for machine SR accuracy to equal human listening accuracy. Currently, continuous speech recognition produces error rates of around 5-10%. The rate of error increases in noisy environments, or in situations involving multiple speakers.

In recent years, most SR research has been concerned with building 'noise robust' recognition models. This includes models that rely on user input to refine and correct recognition errors. Noise can also be dealt with at the source by investigating ideal microphone configurations for noise-free speech capture. The raw captured audio data could also be pre-processed to remove noise.

Deng & Huang (2004) suggest that most of the problems associated with speech recognition arise from the fact that most systems attempt to recognise too much. The paper suggests that future systems should employ semantic knowledge to restrict the range of words that can be recognised in a given context. By reducing the set of alternatives, recognition accuracy will be improved. In our system, a dynamic context-free grammar (CFG) is used to define each of the different commands that users can issue. Certain words are only valid if they form part of a valid command. This approach leads to improved SR accuracy and performance, while allowing a variety of different commands to be issued.

One of the advantages of using speech as an interaction mechanism is that it can be used in conjunction with other input methods. For example, Bers, Miller & Makhoul (1998) describe a system that allows users to order parts for a truck by pointing to an image of the truck with a stylus. The user is able to zoom into parts of the image for a detailed view, and to specify the quantity to order by speaking to the system. The VoiceLog system contains several interoperating modules which deal with speech input recognition; stylus gesture recognition; visual and auditory feedback; and integration/response. The modules are distributed over a computer network to take advantage of parallel processing.

The developers of VoiceLog found that noise-free audio channel was critical in ensuring highly accurate speech recognition. To achieve this, they used a unidirectional ear-mounted, noise-cancelling microphone, to ensure that a clean signal was reproduced.

Although our system is primarily speech-based, the SR component exists as a separate entity from the rest of the system, similar to the architecture described by Bers et al. (1998). This allows the SR module to be updated or replaced as improved technology becomes available, with minimum disruption to the rest of the system.

2.4 Speech Recognition in Augmented Reality

Perhaps the most important application of speech technology would be in the fields of virtual or augmented reality, where intuitive interaction techniques are highly desirable. Goose, Sudarsky & Zhang (2002) describe an AR tool developed to aid in the maintenance of industrial systems. The Speech-Enabled Augmented Reality (SEAR) system makes extensive use of speech recognition and synthesis to provide natural multimodal interaction. In their system, technicians are able to walk around a real-world industrial plant, such as a piping system, and interact with elements such as individual pipes by talking to them. For example, the technician could query the status of a resevoir simply by asking it directly, and the resevoir would 'reply', perhaps stating its current temperature and pressure, using speech synthesis. The users wear head-mounted displays which augment their view of the real world with data about the subsystems being observed.

SEAR makes use of a number of unique visual markers placed at certain positions around a maintenance area, in order to identify different parts of that area. The markers are detected when they enter the technician's field of view – this allows the SEAR software to overlay information specific to the area around that marker, and also to load a speech command set for that area. The software loads and unloads different speech command grammars as the technician moves between different areas. By restricting the range of possible speech commands in each context, speech recognition performance and accuracy is improved. The markers also allow the software to produce positional synthesised speech, using Microsoft DirectX 3D audio streams.

2.5 Augmented Reality Characters

There has also already been some research on using lifelike avatars in a learning environment. Lester James, C, Zettlemoyer Luke, S, Gregaire Joel, P & Bares William, H (1999)describe an approach at implementing and evaluating explanatory lifelike avatars in a 3D learning environment. They found that such avatars are useful as they "can take advantage of humans' inherent propensities to anthropo-

morphize software". For this research, the authors used the CPU City 3D learning environment, a piece of software designed to teach students of computing about how low-level functions in a computer work. This software works by having a 3D representation of computer components, such as the CPU, RAM, and the hard disk. The user can enter typical programming commands such as $F = (9/5) * C + 32$ (the function to convert Celsius to Fahrenheit) and the camera will fly around this 3D environment, showing the user how this command is processed by the computer. The authors created three additional versions of the system, one using a disembodied narrator (simply a voice over), a mute lifelike avatar, and a full-scale explanatory avatar (which is the lifelike avatar, who narrates the current events).

An evaluation was run where users used the baseline system (with no narration or avatars), and two of the other systems. The results showed that participants who interacted with the Agentless 3D world and Explanatory Lifelike Avatar environment unanimously preferred the lifelike avatar, and that participants repeatedly commented about the improved clarity of their understanding by interacting with the environment with the avatar. The task decomposition from a high-level user command to low-level system usable chunks is similar to our system, and the evaluation that users believe that an avatar is a useful learning tool increases the likelihood that our avatars will contribute to the educational content of our system.

Blumberg Bruce, M (1995) also carried out a body of research on autonomous creatures in virtual environments, and described the design and implementation of such creatures. Their motivation stems from creating realistic, yet controllable creatures for these virtual environments. The standard research prior to this work was on strictly behavioral based autonomous creatures, which use various forms of input (usually from the system) to make decisions on which actions to perform. The authors of this paper claim this to be a problem when interacting with humans however, as these creatures can get into such behavior patterns that they actually end up ignoring the user, and just performing tasks they believed to be important. The authors of this paper designed a hybrid directional/behavioural system, which allowed the environment and user actions to be fed into the behavioural processor, and then actions were generated and assigned priorities, so that the character would perform actions in a logical way. This means that the creature (in the application, a dog) could keep itself entertained, but would react to a user when there was one around. This user feedback also allowed for changes in the behavioural system of the animal. Another feature of this method is that it allows the creature to show emotion.

This research offers some useful ideas, such as offering an interface that the behavioral system can call, and then have characters can implement as they see fit. An example would be an action such as "Move To". For a dog character, this may internally call the "Walk" method, while a car character may call the "Drive" method. This means that characters can easily be plugged into the system with no need to rewrite vital parts.

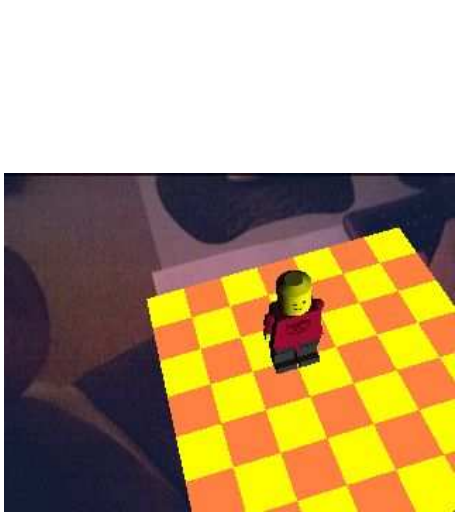
Wray Mike (1999) also carried out research on avatars, with their work being focussed on avatars in LivingSpace. They describe the use of the dead-reckoning technique, combined with Kalman filtering, to provide a fast and low-bandwidth method for animation of realistic human models. The motivation for this was creating avatars for a distributed 3D platform, LivingSpace. Networked users needed to be able to view the avatars and their current animations, and keep the bandwidth consumed by these avatars to a minimum. The avatars were loaded from VRML models, which provide the geometry, as well as information about joints and segments. By breaking down actions into their components these components can then viewed as manipulations on the geometry. To lower the amount of network traffic, each client only needs to receive information about the initial pose, final pose, and time, and can interpolate the action from start to finish using dead-reckoning. Additionally, more packets can be sent during the animation to synchronize all other connected clients.

The idea of allowing the interpolation of movements be dealt with client side is one which was implemented in our research. This allowed us to minimize the amount of work done by particular models when moving.

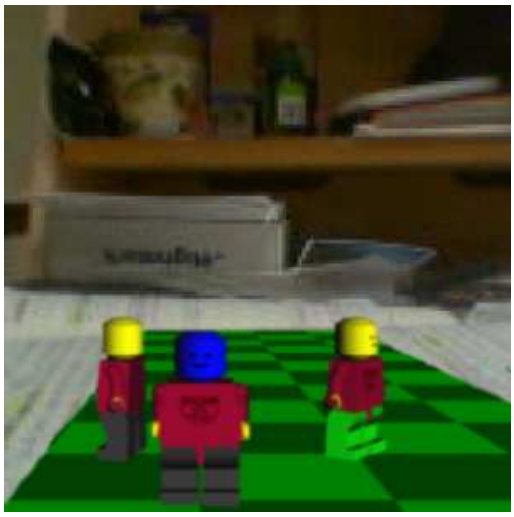
3 Project Goals

Previous research on AR characters detailed above showed their success in involving the user and in encouraging natural interaction with the system. It was hence a main goal of our project that our avatars would prove to be a similar success, being both popular with the user and fun. The target audience for our system also meant that it was important our system be easy to use, with natural interaction.

Since the purpose of our system is to teach novices computer programming, another project goal is to demonstrate that after using our program



(a) In this picture there is one character, on the 'beach' disco floor



(b) In this picture there are three characters, some with parts of their bodies set to different colours, and the 'grass' disco floor.

Figure 1: These screen-shots demonstrate different numbers of characters, different attribute settings and different global variable settings

some computer programming concepts were learnt. It is hoped that the concepts represented in the program can be mapped in a more general sense to the user's mental model of programming. In this way, we expect users to learn what attributes are by setting the value of some attributes of the character. Global variables will also be able to be altered, such as changing the physical environment of the characters. Some examples of how the system would look when attributes and global variables have been altered can be seen in figures 1(a) and 1(b). Method creation and calling will also be available, and some basic sort of iteration as instructions can be repeated a fixed number of times. The Object-Oriented(OO) idea of a class and an instance of this can be represented by the characters themselves, which can be created and deleted. Finally, a basic representation of the idea of `import` statements will be available as users will be able to set which character they are talking to and then call methods or primitive operations, rather than explicitly invoking the character's name before the method each time.

4 Implementation

For implementation purposes, the system was divided into three components as shown in figure 2. This allows the speech recognition module to be decoupled from the animation. The equipment used for development and the details of these components are given below.

4.1 Equipment

Hardware used in the development of this system consisted of a Universal Serial Bus (USB) camera, with the display shown on a computer with a Windows NT environment. However, when implemented in a real-world situation, head-mounted displays would obviously be preferable. During development, speech was processed by the on-board microphone of an ASUS laptop. The problems encountered with this hardware are discussed in section 5.1 below, and it is highly likely that much better performance would be possible with a head-set microphone.

4.2 Speech

In implementing the speech recognition and synthesis component of the system, the Microsoft Speech SDK (*Speech SDK for Windows* 2002) was used. There was a conflict of interests here in that we wanted user input to feel as natural as possible, but the system had to be able to accurately recognise user commands to process them accordingly. This required a small grammar to be defined, although the syntax used followed natural speech as much as possible.

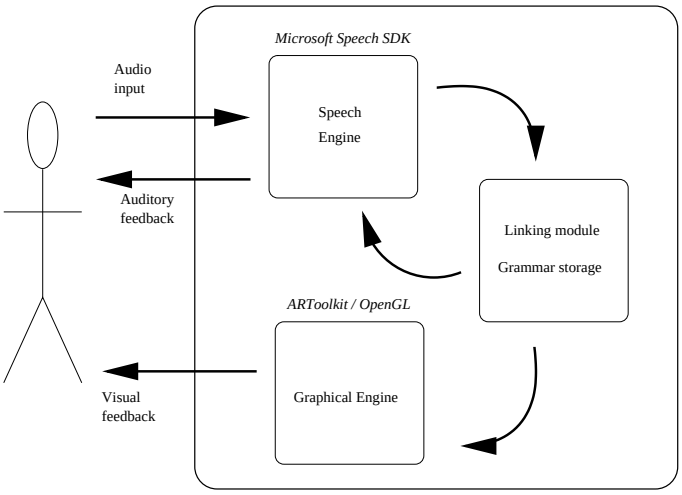


Figure 2: Our system architecture and component communication

Furthermore, speech input could be implemented either through a free dictation mode or a fixed dictionary. Although a fixed dictionary offers much better results, as discussed in section 2.4, this was not always practical. It was part of the requirements that the user be able to label avatars and define methods. Although this decreases accuracy, so long as the system is consistent in its misinterpretation of commands this would not necessarily be a real problem. Another possible approach would be to restrict the available person and method names to a sub-set of a dictionary, such as restricting method names to only verbs.

The grammar had to be wrapped in an XML file[1] to dictate both the structure of commands and a list of all words that could be spoken by the user. This also defined the point at which the mode would switch to free dictation if so desired. Upon learning a new method, this file would have to be changed by the program so as to recognise the new method name. Further detail on this is present in section 4.3.

The recognition part of the system runs in a concurrent thread to the main program execution. This is crucial as otherwise the whole application would halt while voice input was expected. At the same time as the graphics loop is rendering the current picture and preparing the next buffer, the speech recognition component needs to be able to run quietly in the background, and respond to speech input in a non-intrusive manner.

Some speech synthesis was also implemented, something for which the SDK provides good support. Multiple voices were supported, to allow multiple characters to be visible at one time, each with individual voices. This also allows some control over the properties of a particular character as the user can change its voice to suit their preferences. The content of such speech could not only acknowledge user's commands, but provide help if wrong syntax was used, or operations called that either the system did not understand or were not available for the specific character being controlled.

4.3 Linking Module

The linking module is responsible for coordinating the communication between the speech recognition part of our system and the AR module. There are three main communications that take place:

1. The linking module notifies the speech recognition module of the grammar and dictionaries that the user will use to speak to it. This communication will happen at various times when the system is running, as the dictionaries change when the user defines new method and person names.

2. The speech recognition module will inform the linking module of the command the user has just issued.

3. The linking module will use calls to the AR module to create characters and manipulate them based on user-issued commands.

[1] See appendix

4.3.1 Grammar and Dictionaries

The linking module maintains tables containing the dictionaries and grammars used by the system. These define how the user will issue commands to the system, as well as acting as a list of words that the user may not choose for user-defined character and method names. These tables are defined by the system and are read in from a series of customised grammar files. For example, the 'person action' grammar file below specifies on each line firstly an action available for the user to call on the whole character,followed by the attributes of each action.

`gramPersonActions.g`

```
WALK DIRECTION
TURN TURNDIRECTION ANGLE
```

Other grammar files for the system are necessary for the following components:

- Part Actions This is in a similar format to the 'person action' grammar file, except this details actions that can be carried out on a part of a character, such as `lift` or `drop`.
- Program Instructions This contains all program instructions such as creating and deleting new characters, and changing global variables. For each instruction a list of possible attributes is given.
- Grammar Attributes This contains all command attributes mentioned by either the 'part action', 'person action' or 'program instruction' files and specifies all possible values.

The module also maintains the dictionary of character and method names that the user has chosen, which is updated dynamically as the speech module communicates these to the linker.

4.3.2 Communication to Speech Module

To communicate the grammar and dictionaries, including the user defined names, to the speech recognition module, a particular Extensible Markup Language (XML) schema able to be parsed by the speech module was adhered to. This was created by iterating through the tables stored of different types of actions, extracting the relevant information and placing the appropriate xml tags around it.

4.3.3 Communication from Speech Module

The speech module will communicate to the linker the command the user has just issued. The speech module removes any irrelevant words that the user can optionally add to their command, such as "please", so the linker does not need to deal with these. The linker is responsible for translating each user command into calls to the AR module by parsing and interpreting the command. It must create and destroy characters as instructed, and maintain each character object in the correct state. The linker decides for which character an action command is intended, and then makes the appropriate method call on the character or character part object, with the arguments based on the attributes of the command.

4.4 The AR Character

To implement the actual animation, OpenGL was used in conjunction with the ARToolkit. An object-oriented approach was taken, with the main classes defined being those for a `LegoCharacter`, a `BodyPart`, a `Motion` and the `Interface`. These were subclassed as required. A class diagram of the `LegoCharacter` section is shows this in detail in figure 3. Using such a design ensured the system was easier to maintain and debug, but more importantly ensured easy reuse of code and extension of the system. Body parts could hence be mixed easily such that a `LegoCharacter`could be created with a `TeddyBearHead` if so desired. Furthermore, advantages of inheritance in this design meant that if any new body parts were to be added no new motion code would be needed unless specifically desired.

Movement was achieved by attaching a character to a pattern marker, then passing in the initial and final positions, and interpolating between these positions. Different methods of interpolation were tried with varying success, as discussed under section 5.1. If the pattern is either occluded or moved beyond the field of view, the character will disappear. When the pattern is again recognised, the character will be reset to the centre of the pattern. An example of our character attached to a pattern marker can be seen in figure 4.

5 Discussion

5.1 Implementation Challenges

There were a few implementation difficulties encountered during the development of this system.

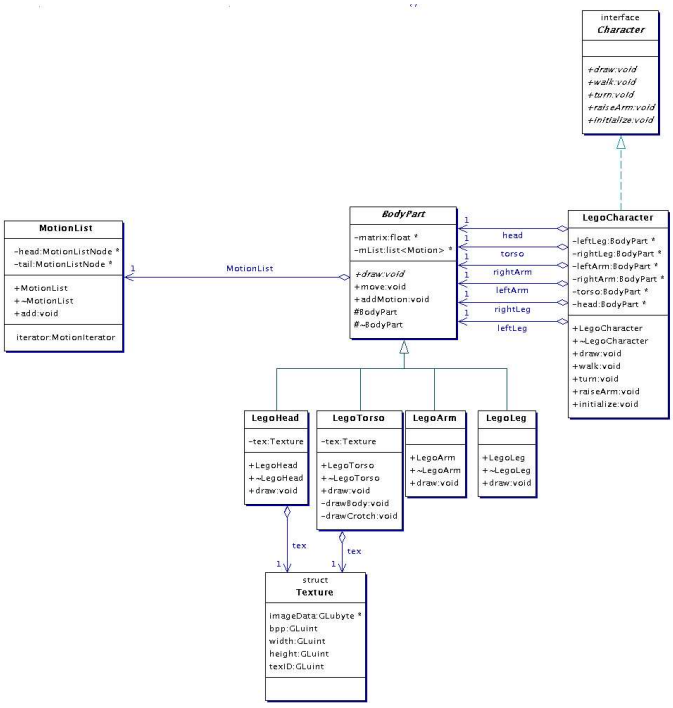


Figure 3: Class diagram of the character and motion implementation

The first of these was a hardware issue with the speech recognition module. In the initial speech recognition testing phase it was very difficult to get the system to correctly identify the words spoken. Speech recognition systems typically require some level of user practice or a very controlled environment to be very effective. As this system was designed for children, and is intended to be easy to use the only viable option for improving the success of recognition was to set up the environment so it could be as close to the ideal environment as possible. This involved making hardware adjustments, for example to the recording level to which the microphone was set, and eliminating background noise. Ideally more expensive equipment that was custom designed for speech recognition could alleviate the problems in this area. Head-set microphones in particular would be useful, as these would ensure that any sound accepted was from the direction of the user, rather than accepting input from any direction, and ensure the microphone would stay close to the user.

The other aspect of the project in which implementation challenges arose was in the OpenGL implementation of the animated character. The motion of the character was originally coded using linear interpolation. This was designed so that only the initial and final positions needed to be specified. However, this technique yielded some problems including the motion being very disjointed and jerky. It also had some accuracy problems, for example an object told to rotate clockwise 90 degrees four times would not end up facing exactly the same way as its original orientation. This problem was solved by altering the interpolation technique for the motion to a matrix based implementation. To further improve the situation additional constructs were introduced. The progress through a motion was tracked and the remaining distance or angle the motion was to pass through was calculated by this.

The ARToolkit itself proved easy to attach to the system. However, only single pattern recognition was implemented. This means that a character is always attached to the same pattern, and is only visible while that pattern is in frame. Ideally, a series of patterns could be used, and the character would be able to move from one to another.

5.2 Exploratory Evaluation

Although the limited time available for this project meant that it became unfeasible to implement the full system, different components of the system were available in prototype form for some prelim-

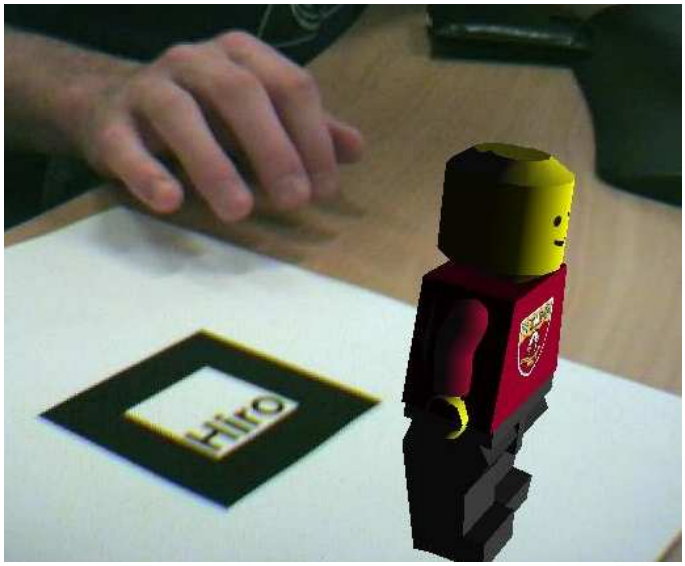


Figure 4: This shows an ARToolkit pattern marker with the associated character from our system

inary explorative user evaluation. Users were asked to define several programming concepts as part of a pre-test, namely global variables, attributes, methods, loops, classes, constructors and import statements. After this, a short explanation of a concept was given, followed by an opportunity to complete a task related to this by issuing verbal commands from a small subset of the entire envisaged grammar. The animated character was then manipulated according to these. Once every concept had been taught, users sat a post-test. This required them to issue a command in accordance with a given instruction such as "Change a global variable in this scene". They were also asked to explain the relevant programming concept. Finally, they were asked to comment on how helpful they found the system, how much they felt they learnt, how easy the system was to use and how much they enjoyed the experience.

Results from this were very encouraging. As expected, the participants were unable to define any of the required programming concepts in the pre-test. However, after the concepts were explained and a related example task provided for them to work through, the post-test results were pleasing. All participants demonstrated their new knowledge both by issuing the correct verbal command associated with the concept and by giving an explanation in more abstract terms of what the concept meant. Although this was an informal pilot study, it still shows the potential here for educative benefits. Furthermore, participants were very enthusiastic about the interface, describing it as easy to use, fun and a good learning tool.

5.3 Future Work

There is a lot of scope for carrying out further research related to this system. Evaluation of the system is one aspect that could be more deeply investigated. Firstly, there could be a more thorough usuability evaluation with participants from our target user group, particularly children. It would be useful to videotape the experiment to gain an insight as to how successful the system is at providing a fun and natural interface for children to use. Supplementing this, the participants could be asked to provide subjective responses to the system. On the basis of results from such an evaluation the system could be modified to ensure it does achieve our goal of being accessible and fun.

A formal evaluation of the educational value of this system would greatly increase the significance of the research. It would be desirable to see if students using the system over time had a significant learning advantage over children using more conventional methods of learning programming concepts. In addition, it would be interesting to compare this research to other similarly less traditional educational tools for children learning to program, such as ToonTalk(Ken 2004), which is a standard PC application that teaches students using fun animations.

There are also some extensions that could be

made to the system we have developed. The introduction of more programming metaphors would help align the research more closely to a programming curriculum. As this system is primarily aimed as an introduction to OO programming and we already have the animated characters as objects, a next step would be to have interactions between different characters. Allowing one character to manipulate another by making calls to its methods would be a good introduction to how objects interact in an OO program. Other programming concepts could also be incorporated, such as introducing new objects to represent different classes, and even perhaps introducing some inheritance and polymorphism here.

Another extension that would be interesting from the AR perspective would be to include the use of gestures when controlling the characters. This could be done by applying some of the technologies already developed in manipulating AR objects, such as using a paddle. It would allow a richer variety of interactions with the characters and could also provide a further means of interaction between two characters. For example, commands could trigger different actions depending on the gestures used. This would mean the user could refer to a certain object as 'this' object by pointing to it. It would also create alternative methods for the creation and destruction of characters.

6 Conclusion

It can hence be seen that this application of AR technology is a viable one, and a full extended version could prove to be useful for institutes offering introductory programming courses. The speech component can recognise user commands as per the grammar defined to a reasonable accuracy, and one that would only be improved with better microphone technology. However, there are still significant problems with the free dictation mode. Speech synthesis is available for the characters to interact with the user. The animation has been implemented for a subset of possible commands, and limited evaluation hence undertaken. Further work could see the system extended to cover more programming concepts, and a formal empirical evaluation performed. However, even the basic system components as currently implemented still proved popular with users, and more importantly it could be seen that it is possible to learn programming concepts by manipulating the 3D character and his environment. This should promote further exploration into how AR can facilitate education in a variety of domains. AR is becoming much more accessible as the technology decreases in expense and head-mounted displays become more comfortable and hence more accepted by the general public. The inherent benefits of AR hence both could and should be exploited in the future to develop more educational tools.

References

Azuma, R. & Bishop, G. (1994), Improving static and dynamic registration in an optical see-through hmd, *in* 'International Conference on Computer Graphics and Interactive Techniques', pp. 197–204.

Bers, J., Miller, S. & Makhoul, J. (1998), 'Designing conversational interfaces with multimodal interaction', *DARPA Workshop on Broadcast News Understanding Systems* .

Blumberg Bruce, M, G. (1995), Multi-level direction of autonomous creatures for real-time virtual-environments, *in* 'Proceedings of the 22nd annual conference on computer graphics and interactive tecchniques'.

Bricken, M. (1991), 'Virtual reality learning environments: potentials and challenges', *ACM SIGGRAPH Computer Graphics* **25**(3), 178–184.

Deng, L. & Huang, X. (2004), 'Challenges in adopting speech recognition', *Communications of the ACM* **46**(1).

Goose, S., Sudarsky, S. & Zhang, X. (2002), 'Speech-enabled augmented reality supporting mobile industrial maintenance', *IEEE Transactions on Pervasive Computing* **2**(1).

H, K. (2002), Construct3d: An augmented reality application for mathematics and geometry education, *in* 'Proceedings of the 10th ACM International Conference on Multimedia'.

Ken, K. (2004), The child-engineering of arithmetic in toontalk, *in* 'Proceedings of the 2004 Conference on Interaction Design and Children: building a community'.

Lester James, C, Zettlemoyer Luke, S, Gregaire Joel, P & Bares William, H (1999), Explanatory lifelike avatars: performing user-centered tasks in 3d learning

environments, *in* ‘Proceedings of the third annual conference on autonomous agents’.

Mayer, R. E. (1981), ‘The psychology of how novices learn computer programming’, *ACM Computing Surveys* **13**(1), 121–141.

Powell, N., Moore, D., Gray, J., Finlay, J. & Reaney, J. (2003), Dyslexia and learning computer programming, *in* ‘LTSN-ICS Annual Conference Proceedings’, Vol. 3, LTSN-ICS, Galway.

Ramalingam, V., LaBelle, D. & Wiedenbeck, S. (2004), Self-efficacy and mental models in learning to program, *in* ‘Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education’, pp. 171–175.

Speech SDK for Windows (2002), http://www.microsoft.com/speech/download/sdk51/.

Wray Mike, B. (1999), Avatars in livingspace, *in* ‘Proceedings of the fourth symposium on Virtual Reality modeling language’.

Appendix 1: XML file generated for use by speech component

```
<GRAMMAR LANGID="409">
    <DEFINE>
        <ID NAME="VID_Counter" VAL="1"/>
        <ID NAME="VID_Place" VAL="253"/>
        <ID NAME="VID_Navigation" VAL="254"/>
        <ID NAME="COMMAND" VAL="211"/>
        <ID NAME="ACTION" VAL="212"/>
        <ID NAME="PERSON-NAME" VAL="213"/>
        <ID NAME="METHOD-CALL" VAL="214"/>
        <ID NAME="INSTRUCTION" VAL="215"/>
        <ID NAME="PART-INSTRUCTION" VAL="216"/>
        <ID NAME="PERSON-INSTRUCTION" VAL="217"/>
        <ID NAME="METHOD-NAME" VAL="218"/>
        <ID NAME="BODY-PART" VAL="219"/>
        <ID NAME="LIFT-AMOUNT" VAL="220"/>
        <ID NAME="BODY-PART-DIR" VAL="221"/>
        <ID NAME="OTHER-AMOUNT" VAL="222"/>
        <ID NAME="MOVEMENT-DIR" VAL="223"/>
        <ID NAME="ANGLE-DIR" VAL="224"/>
        <ID NAME="ANGLE-AMOUNT" VAL="225"/>
        <ID NAME="WHICH-SIDE" VAL="226"/>
        <ID NAME="PART" VAL="227"/>
        <ID NAME="SINGLE-PART" VAL="228"/>
        <ID NAME="INT-NUM" VAL="229"/>
        <ID NAME="INT-ANGLE" VAL="230"/>
        <ID NAME="LEARN-METHOD" VAL="231" />
        <ID NAME=''PROGINST'' VAL=''232'' />
        <ID NAME=''PROGINST'' VAL=''233''' />
        <ID NAME=''NEWCHAR'' VAL=''234'' />
        <ID NAME=''DELCHAR'' VAL=''235''' />
        <ID NAME=''COLOUR'' VAL=''236'' />

    </DEFINE>
        <RULE ID="COMMAND" TOPLEVEL="ACTIVE">
        <P>
        <L>
                <RULEREF REFID=''PROGINST''/>
                <RULEREF REFID="ACTION" />
        </L>
        </P>
        </RULE>

        <RULE ID=''PROGINST''/>
                <L>
                <P>environment set
                        <L>
                                <RULEREF REFID="LOCATION" />
                        </L>
                </P>
                <P>
                            <RULEREF REFID="LEARN-METHOD" />
                </P>
```

```
        <P>
              <RULEREF REFID="NEWCHAR" />
        </P>
          <P>
              <RULEREF REFID="DELCHAR" />
        </P>
        <P>set
                    <RULEREF REFID="BODY-PART" />
                    <RULEREF REFID=''COLOUR'' />
        </P>
        <P>hi
                       <RULEREF REFID=''PERSON-NAME'' />
        </P>
        </L>
</RULE>

<RULE ID="ACTION">
        <O><RULEREF REFID="PERSON-NAME" /></O>
        <L>
                  <RULEREF REFID="METHOD-CALL" />
                  <RULEREF REFID="INSTRUCTION" />
        </L>
</RULE>

<RULE ID="INSTRUCTION">
<P>
        <L>
                <RULEREF REFID="PART-INSTRUCTION" />
                <RULEREF REFID="PERSON-INSTRUCTION" />
        </L>
                <O><RULEREF REFID="OTHER-AMOUNT" /></O>
</P>
</RULE>

<RULE ID="METHOD-CALL">
        <P>do</P>
        <RULEREF REFID="METHOD-NAME" />
</RULE>

<!-- dynamic rules: names of characters and methods -->
<RULE ID="PERSON-NAME">
        <P>nobody</P>
</RULE>

<RULE ID="METHOD-NAME">
        <P>nothing</P>
</RULE>

<!-- end of dynamic rules -->

<RULE ID="PART-INSTRUCTION">
        <L>
                <P>lift
                        <RULEREF REFID="BODY-PART" />
```

```
                        <RULEREF REFID="LIFT-AMOUNT" />
                        <RULEREF REFID="BODY-PART-DIR" />
                        degrees
                </P>
        <P>DROP
                <RULEREF REFID="BODYPART" />
                <RULEREF REFID="LIFTAMOUNT" />
                degrees
        </P>
</L>
</RULE>

<RULE ID="PERSON-INSTRUCTION">
        <L>
                <P>
                        walk
                        <RULEREF REFID="MOVEMENT-DIR" />
                </P>
                <P>
                        turn
                        <RULEREF REFID="ANGLE-DIR" />
                        <RULEREF REFID="ANGLE-AMOUNT" />
                        degrees
                </P>
        </L>
</RULE>

<RULE ID="BODY-PART">
        <L>
                <P>
                        <RULEREF REFID="WHICH-SIDE" />
                        <RULEREF REFID="PART" />
                </P>
                <RULEREF REFID="SINGLE-PART" />
        </L>
</RULE>

<RULE ID="WHICH-SIDE">
        <L>
                <P>left</P>
                <P>right</P>
        </L>
</RULE>

<RULE ID="PART">
        <L>
                <P>leg</P>
                <P>arm</P>
        </L>
</RULE>

<RULE ID="SINGLE-PART">
        <P>head</P>
</RULE>
```

```
<RULE ID="LIFT-AMOUNT">
        <L>
                <P>half</P>
                <P>full</P>
        </L>
</RULE>

<RULE ID="BODY-PART-DIR">
        <P>to</P>
        <L>
                <P>front</P>
                <P>back</P>
                <P>left</P>
                <P>right</P>
        </L>
</RULE>

<RULE ID="OTHER-AMOUNT">
        <RULEREF REFID="INT-NUM" />
        <P>times</P>
</RULE>

<RULE ID="INT-NUM">
        <L>
                <P>one</P>
                <P>two</P>
                <P>three</P>
                <P>four</P>
                <P>five</P>
                <P>six</P>
                <P>seven</P>
                <P>eight</P>
                <P>nine</P>
                <P>ten</P>
        </L>
</RULE>

<RULE ID="MOVEMENT-DIR">
        <L>
                <P>forward</P>
                <P>backward</P>
                <P>left</P>
                <P>right</P>
        </L>
</RULE>

<RULE ID="ANGLE-AMOUNT">
        <RULEREF REFID="INT-ANGLE" />
        <P>degrees</P>
</RULE>

<RULE ID="INT-ANGLE">
        <L>
```

```
                        <P>forty five</P>
                        <P>ninety</P>
                        <P>one hundred and thirty five</P>
                        <P>one hundred and eighty</P>
                        <P>two hundred and twenty five</P>
                        <P>two hundred and seventy</P>
                        <P>three hundred and fifteen</P>
                        <P>three hundred and sixty</P>
                </L>
        </RULE>

        <RULE ID="ANGLE-DIR">
                <L>
                        <P>right</P>
                        <P>left</P>
                </L>
        </RULE>
         <RULE ID="LOCATION">
                <L>
                        <P>beach</P>
                        <P>grass</P>
                        <P>chess</P>
                </L>
        </RULE>
         <RULE ID="COLOUR">
                <L>
                        <P>red</P>
                        <P>yellow</P>
                        <P>blue</P>
                </L>
        </RULE>
        <RULE ID="NEWCHAR">
                 <P>new character</P>
                <DICTATION MIN="1" MAX="1" />
        </RULE>
        <RULE ID="DELCHAR">
                <P>bye
                         <RULEREF REFID="PERSON-NAME" />
                </P>
        </RULE>
        <RULE ID="LEARN-METHOD">
                <P>learn method</P>
                <DICTATION MIN="1" MAX="1" />
                <P>learned</P>
        </RULE>

</GRAMMAR>
```