

2003

## Online avatar based interactions

Han Wei Koay  
*Edith Cowan University*

Follow this and additional works at: [https://ro.ecu.edu.au/theses\\_hons](https://ro.ecu.edu.au/theses_hons)



Part of the [Communication Technology and New Media Commons](#)

---

### Recommended Citation

Koay, H. W. (2003). *Online avatar based interactions*. [https://ro.ecu.edu.au/theses\\_hons/573](https://ro.ecu.edu.au/theses_hons/573)

This Thesis is posted at Research Online.  
[https://ro.ecu.edu.au/theses\\_hons/573](https://ro.ecu.edu.au/theses_hons/573)

# Edith Cowan University

## Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Online avatar based interactions

Han Wei Koay

Bachelor of Communications  
Interactive Multimedia, Honours

School of Communication and Multimedia

19 May 2003

## USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

## Abstract

---

The gridWorld project attempts to utilize 3D to develop an online multi-user visual chat system. GridWorld address ideas of how conversations in a virtual environment can be facilitated and enhanced by an abstract visual interface design. The visual interface was developed from research and examination of existing ideas, methodologies and application for development of user-embodiment, chat/virtual space, and interface useability towards the visualization of communication.

## Declaration

---

I certify that this thesis does not, to the best of my knowledge and belief:

- (i) incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education;
- (ii) contain any material previously published or written by another person except where due reference is made in the text; or
- (iii) contain any defamatory material

## Acknowledgments

---

Thanks to God who helped me get through 4 years of university. Thanks to Dr Arshad Omari for his advice and guidance in writing this thesis. Thanks good to the work done by Judith Donath and her "Social Media Group" for providing the inspiration for gridWorld. Thanks to all the SCAM academic and administration staff that provided the tools and support for me get to where I am today.

# Contents

---

<b>Abstract</b>	<b>1</b>
<b>Declaration</b>	<b>1</b>
<b>Acknowledgments</b>	<b>2</b>
<b>Contents</b>	<b>3</b>
<b>1.0. Preface</b>	<b>5</b>
<b>2.0. Research Questions</b>	<b>6</b>
<b>3.0. Introduction</b>	<b>7</b>
3.2. Project Aims	12
3.3. Significance	12
<b>4.0. Literature Review</b>	<b>17</b>
4.1. Existing related studies and projects:	17
4.2. The Communication Channel	23
<b>5.0 The design and implementation of gridWorld:</b>	<b>28</b>
5.1 Designing User Representation:	29
5.2 User representation in gridWorld	32
5.3 Building gridWorld	33
<b>6.0. Multi-channelled nonverbal communication in gridWorld</b>	<b>38</b>
6.1. Paralanguage	38
6.2 Objects	39
6.3. Actions	40
6.4. Appearance	42

6.5. Space and location	44
<b>7.0. Post development / User feedback</b>	<b>46</b>
7.1. QUIS – Questionnaire for User Interface Satisfaction	46
7.2. Results of QUIS Questionnaire and user feedback	46
7.3. Analysis of system usability	52
7.4. Analysis of effectiveness of nonverbal metaphors	54
7.5. Future Improvements And Possibilities	56
<b>8.0. Conclusion</b>	<b>59</b>
<b>Reference:</b>	<b>60</b>
<b>Bibliography:</b>	<b>62</b>
<b>Appendix 1 – QUIS Sample Questionnaire</b>	<b>64</b>
<b>Questionnaire for User Interface Satisfaction</b>	<b>65</b>
<b>Appendix 2 – User Instructions</b>	<b>69</b>
<b>Appendix 3 – Lingo code</b>	<b>70</b>
A2.1. “Script” Cast: Behaviour Scripts	71
A2.2. “Script” Cast: Parent Scripts	84
A2.3. “Script” Cast: Movie Scripts	112



## 1.0. Preface

---

GridWorld is currently held on a private server within Edith Cowan University, School of Communication and Multimedia Sciences. Users can access the gridWorld project homepage through <http://malevich.dyndns.org/~hanwei>.

## 2.0. Research Questions

---

Research in the gridWorld project centred on developing a system of visual representation of verbal and nonverbal communication between users in an online environment. As such, research needed to first identify what verbal, visual and other communication cues are useful to facilitate and enhance communication and interaction in an online avatar-based chat environment. Based on these findings, research next needed to identify what visual metaphors are appropriate for visualizing these communication/cues in an avatar-based chat environment.

Based on these assumptions, the project's research questions are as follows:

- What verbal, visual and/or other communication cues are useful to facilitate and enhance communication and interaction in an online avatar-based chat environment?
- What visual metaphors are appropriate for visualizing communication in an avatar-based chat environment?

## 3.0. Introduction

### 3.1. Background

#### *Chatting online:*

Online chat systems are conversational interfaces that allow multiple users to hold real-time dialogues with each other in an online space. They harness the Internet's property of connectivity to allow numerous people to communicate and interact regardless of time and location through interfaces such as chat systems.

Chat systems can be generally grouped into two categories:

- **Text-based environments:** in which communication is carried out in an almost exclusively word based environment.
- **Graphical environments:** in which text-based communication is heavily augmented with visual cues. These visual cues tend to revolve around the user representation as an avatar.

#### *Text-based chat environments:*

In text-based chat environments conversations are held in an almost exclusively word-based environment. Identity and utterances are displayed together in a list, ordered by time of post, allowing users to gather and converse in real time (figure 1). More advance text-chats such as MSN Messenger (figure 2) allow users to add small graphics in their text to express emotions or actions.

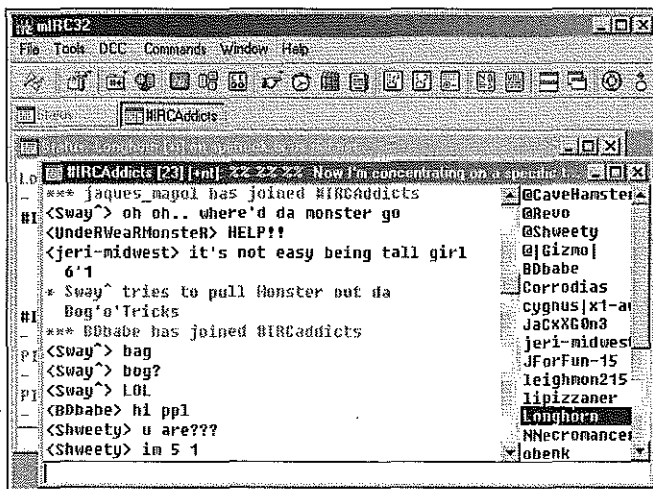


Figure 1 - mIRC ([www.mirc.com](http://www.mirc.com))

Though simple, traditional text-based chats are still very popular. The simplicity of systems such as mIRC, MNS Messenger, Yahoo Chat, and others “provide a rich environment for social interaction (where users can)

gather and converse easily and fluidly” (Spiegel, 2001). All a user need do is type a message, and it appears on screen for all to read.

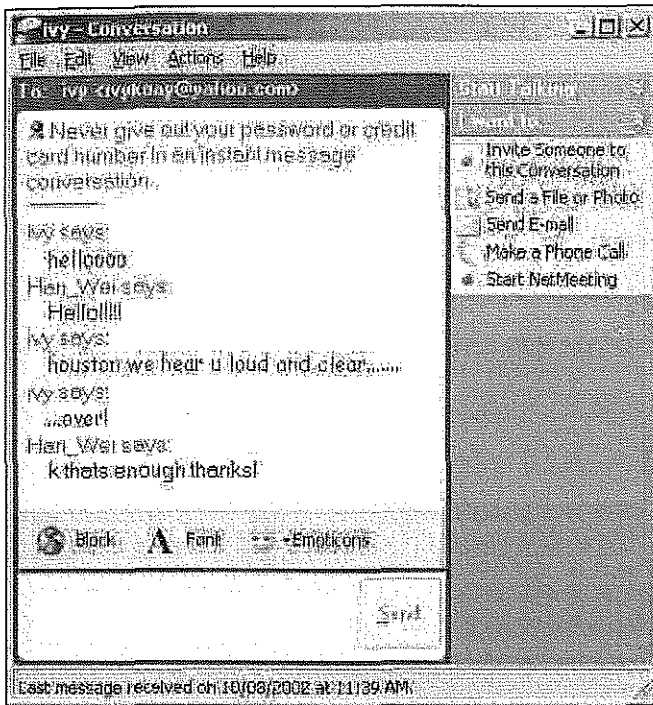


Figure 2 – MSN Messenger

Baker quotes Culnan and Markus (1987) who dubbed this phenomenon as the “cues-filtered-out” approach, where “substituting technology-mediated for face-to-face communication will result in predictable changes in intra-personal and inter-personal variables” – namely that in a purely text-based chat, we are blind to our social surroundings (being incapable of ‘reading’ other people/groups) due to our inability to ‘see’ them.

#### *Non-verbal communication - Social/Visual Cues:*

When socializing in the real world, people rely on both verbal and nonverbal communication in their interactions with one another. When participating or observing a conversation, we are constantly sending and receiving nonverbal information to establish identity, meaning, and intent, such as cues for establishing turn-taking or social behaviour. Non-verbal communication thus allows us to adapt and coordinate ourselves to everyday social interchange.

Blake and Haroldson define non-verbal communication as “the transfer of meaning involving the absence of sound or sound representations” (1975). Motion, gestures,

While effective, text-chats face several limitations due to their purely text-based interface. As all communication is text-based, information retrieval is never at-a-glance, but requires the user to scan through lines of text to identify or differentiate users and their postings. Additionally, a completely text-based interaction “lacks the nonverbal social cues used to interpret visual statements” as identified by Baker (1990).

facial expressions, voice-quality/inflections, physical space, smell, touch, and other cues can communicate much meaningful information independently or inconjunction with each other. Goffman has differentiated these cues as being those that are “given” and those that are “given-off” (Spiegel, 2000). Cues that are *given* are explicit expressions of social behaviours, such as nodding in agreement or waving to get attention. Clues that are *given-off* are those that are unconscious or involuntarily, such how we stand or how we look. These are said to form the “back-channels” of social interactions that allow us to ‘read’ a person (or group) and make assumptions of their activity and expectations. Though non-verbal communication can encompass all senses, it is the non-verbal visual cues that are of interest in this research.

Lee (2001) notes that the goals of visual cues are to “augment real-life conversations with additional detail or contextual hints” (p. 13). Though limited, users of traditional text-based chat are still able to communicate nonverbal social expressions. Emotions and actions can be expressed and understood through use of *emoticons* and *social acronyms*. ‘Emoticons’ take advantage of ASCII text characters to produce various expressive faces for emotional responses (e.g. ;- ) for ‘winking’ or :- ( for ‘sadness’). Likewise, social acronyms like ‘LOL’ (laughs-out-loud) and ‘WYSIWYG’ (What-you-see-is-what-you-get) can be used to express actions or phrases. More advanced text-chats allow for small graphics (smiley faces, roses, etc) to supplement text dialogues, these have grown out of the replacement of emoticons made possible by newer software technologies.

However the ability to express the “back-channel” visual cues in text-chats is virtually nonexistent. Lacking visual cues, users cannot easily identify who is participating in any number of group conversation, where the attention is directed, or who is present (Donath, 1997). Spiegel highlights that users are functionally invisible to active users until they chat, and that constant attention is needed to keep track of the activity of a channel. A text-based interface cannot express the ‘shape’ of an online community. Spiegel states:

“we cannot see a person’s facial reaction, nor can we see those visual qualities that a person evolves over time, such as their comfort or experience within an environment or how involved they have been in a conversation .... [Offline] we

can, by glancing around a party, see who knows whom, determine which conversations are active and lively, and pick out who has a history with the group. Online, we have little or no such facility.” (2000, p. 4)

*Graphical Chats:*

The term Graphical chat has become commonly associated with those systems that employ avatars moving and interacting in 2D or 3D virtual environments. Ranging from simple 2D characters or pictures to 3D models, avatars are (often) customisable visual representations of users in a virtual chat space. They act as a virtual “body” providing a method of denoting their user’s presence and location, adding an extra

dimension of interaction by giving users the ability to see and approach each other.

Though verbal communication is still done through a text-based channel, utterances are displayed next to their speaker’s avatar much like a ‘speech-bubble’, allowing messages to be more easily associated to or distinguished from conversing users.

Avatar systems, offer a potential channel for expressing the missing visual cues that were previously omitted in traditional text-based chats. Through visual characteristics, movements and animations of the avatars, unspoken information of size, shape, movement, pictographs/

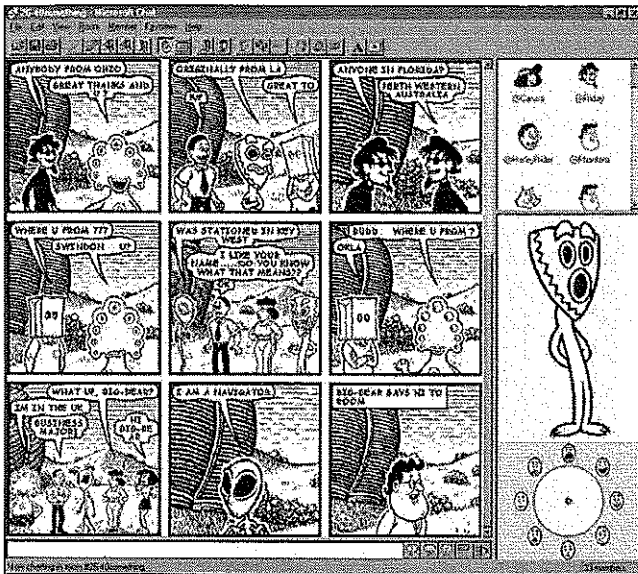


Figure 3 - Comic Chat (Microsoft, 1996).

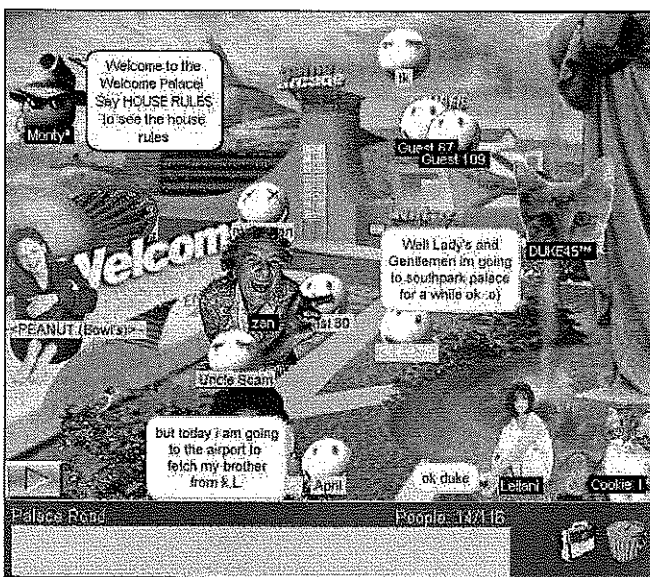


Figure 4 - The Palace

images, colour, etc can communicate presence, activity, availability, mood, status, location, identity, capabilities and many other information (Benford, Bowers, Fahlén, Greenhalgh, & Snowdon, c. 1995) normally available in an offline situation. Additionally, the use of iconic graphics helps to alleviate information overload while adding some aesthetic value to the chatting experience.

Through combining social expressions and visual cues, avatar systems have the potential to enhance the effectiveness and enjoyment of online chatting.

### 3.2. Project Aims

The gridWorld project seeks to build a graphical 3D avatar chat system. The prototype system will seek to address deficiencies identified in current graphical chat systems.

The research has four aims:

- 1) To identify what visual cues and social behaviour expressions are needed for basic conversational communication and interaction between users in 3D avatar based virtual environments through examination of current research and practice
- 2) To develop a three-dimensional avatar-based chat prototype incorporating the visual cues and expressions identified by the research.
- 3) To evaluate the system's usability and effectiveness at facilitating online chatting.
- 4) To evaluate the system's success at implementing the non-verbal elements identified.

### 3.3. Significance

Current graphical chats systems can be roughly categorised as either two-dimensional or three-dimensional. Two-dimensional chats, such as The Palace (figure 4) incorporate a 2-dimensional chat space within which iconic or graphical avatars congregate and communicate within a single or series of chat screens. Three-dimensional chats like Active Worlds (figure 5) feature 3D worlds in which users view, explore, and interact in the virtual world from a first or third person view of their representative avatars.

#### *Shortcomings of 2D Graphical Chats:*

Two-dimensional graphical chats such as The Palace and Comic Chat suffer from the problem of limited on screen 'real-estate'. As more users join a channel, more space is consumed to display their avatars. Similarly, when a user "speaks", additional chat space is taken up to display their utterances' message text. If user numbers are not limited, this will result in a cluttered and unintelligible visual kaleidoscope of graphics and text, overlaying and obscuring each other.



Additionally, and more importantly, though the user is able to change or animate their representations, there is little direct or continuous connection between user and avatar. Systems such as Comic Chat or the Palace offer users little or no control over the movements of their avatars or allow only for discontinuous motion within the chat space (Spiegel, 2001).

### Shortcomings of Graphical 3D Chats:

Popular 3D space graphical chat approaches, though aesthetically spectacular, are often inefficient, superficial and tend to be distracting to the task of chatting itself. (Benford et al., c. 1995; Donath, 1997; Mania & Chalmers, 1998; Miller, Mitchell, Eva, & Wood, c. 2000) The shortcomings of such a system will be discussed using Active Worlds as an example of an existing graphical chat system.

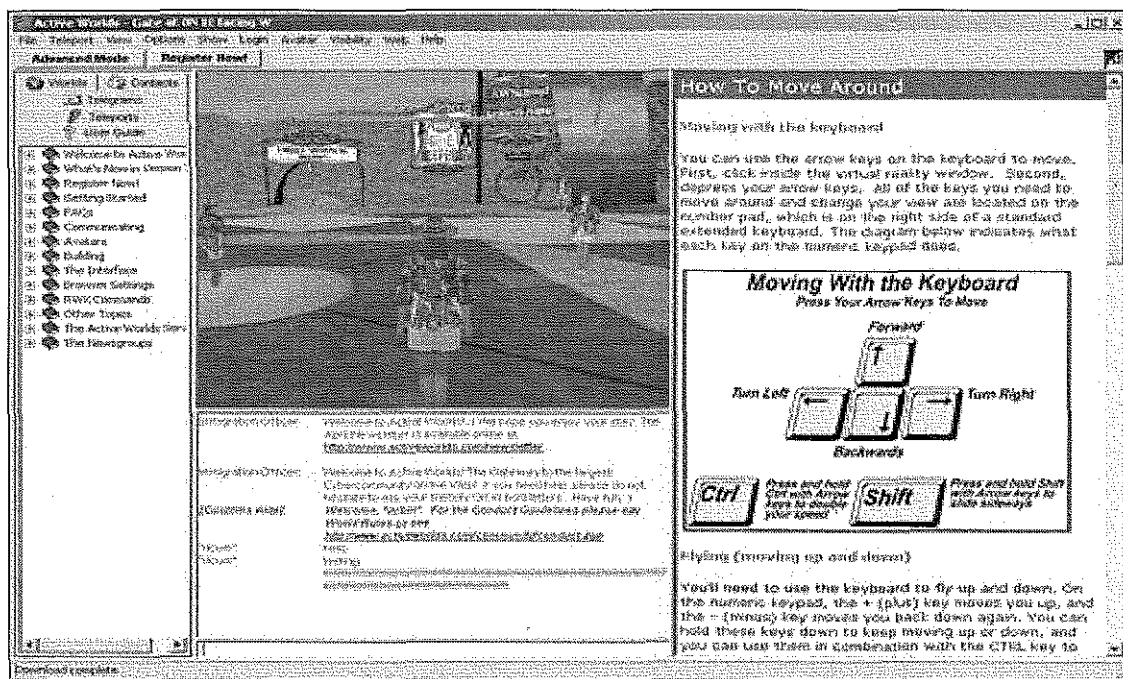


Figure 5 - Active Worlds Browser Interface ([www.activeworld.com](http://www.activeworld.com))

Active Worlds is a reflection of the current state-of-art in graphical chat systems. Users access Active Worlds through the AWB (Active Worlds Browser), navigating from a first or third person point of view from their avatar. Avatar models are typically articulated humanoids with customisable gender, ethnicity and clothing. The user communicates by typing into a chat field and text messages are displayed in the text window as well as on screen above the avatar, visible to the 12 nearest users. A selection of predefined emotions and actions ('happy', 'angry', 'wave', etc) are

available for expressing nonverbal comments, as well as random automatic avatar actions during moments of user inactivity (such as looking at his/her watch).

*An example of avatar interactions in Active Worlds:*

"Paul walks up to Susan who stands there staring blankly out into space. "Hello Susan, how are you?" Susan looks at her watch as she replies "Paul! Great to see you! I'm fine, how have you been?" Paul returns the stare and without twitching a limb he exclaims, "Real Life sucks, I don't think I'm going back there :)". Susan looks at her watch. Paul continues, "I mean, out there you can't just walk up to a random person and start a conversation". Susan looks at her watch. Karen says "Hi". While Paul rotates a full circle looking for Karen, Susan replies, "I know what you mean". Karen says, "So what do you guys think about this place?". Karen is over by the fountain, waving. Susan looks blankly at Paul as she says, "I think it is great to actually see the people you are talking to!". Paul is stiff. Karen is waving. Susan looks at her watch."

Excerpt taken from "Autonomous Communicative Behaviours in Avatars"

Page 15, (Vilhjálmsón, 1997)

*Behaviours:*

It is the attempt to emulate the 'real' that is the major shortcoming of systems like Active Worlds. In such systems, users must select from menus of animated sequences or emotional representations to express social behaviours. However, many visual cues important to the real-life conversations are those that are automatic and/or involuntary and so impossible to explicitly select. Additionally, Vilhjálmsón (1997) explains that "lively emotional expressions in interaction is in vain if (the) mechanisms for establishing and maintaining mutual focus and attention are not in place"(p. 16). A lone person standing stiff, staring fixedly into space with a broad smile on his face would be responded with caution rather than welcome.

Lee (2001) identifies that such behaviours in these systems ('waving' or 'looking-at-the-watch') are used more for "(visual) maintenance" than for contextual hints. Serving no purpose in enhancing the conversation, they are "expectations which must be satisfied...(to) make the avatar's appearance acceptable and believable" (p. 10). The use of behaviours for maintenance can also be risky, as a wrong action or expression can "easily and efficiently communicate highly misleading social (and visual) cues." (Donath, 1997)

*Articulated Modelling:*

Literature examining collaborative interfaces (Benford et al., c. 1995; Donath, 1997; Mania & Chalmers, 1998; Miller et al., c. 2000) express concerns for the efficiencies of such systems in terms of their user embodiment, movement/behaviour/action control and chatting. In their discussion of the efficiency of user embodiment Benford et al. note:

“The attempt to reproduce the human physical form in as full detail as possible may in fact be wasteful (where) more abstract approaches which reflect the above issues in simple ways may be more appropriate.”

(c. 1995)

To fully render a three-dimensional human model as well as a detailed virtual landscape would require much computing and hardware resources. Additionally, Lee observes that the choice of human-like avatars brings with it pre-existing stereotypes and expectations that requires additional processing for unnecessary maintenance behaviours to support (p. 10), thus consuming more user resources. This can be seen as wasteful, as even the most articulate avatars tend to become secondary when the user is engrossed in conversation:

“Although (avatar) systems have now become graphically rich...graphics are there simply to provide fancy scenery...while the act of communication is still carried out through a single word-based channel.” (Vilhjálmsón, 1997, p. 2)

#### *Dual-Control:*

Problems and deficiencies with this approach occur in trying to control the avatars' movements and actions during a conversation. Control of avatar actions and behaviours in Active Worlds requires users to break from chatting to activate the various animation sequences or expressive looks. The user's attention is thus divided between chatting and animating the avatar; or else the avatar will stand motionless or initiate some random action. The result is a dissociative effect similar to that of conventional two-dimensional chats, where “motions do not give the impression that you are watching a real person interacting” (Spiegel, 2001, p. 49).

Mania and Chalmers (1998) found that most interfaces, control of an avatar's movements and expressions are often non-intuitive and disjointed from chatting

operations. Not only can this become burdensome, but also the disjointed relationship between body and conversation can lead to “misleading or even conflicting visual cues to others” (Vilhjálmsón, 1997, p. 15). Mania and Chalmers argue that avatar movements and expressions should build towards the chat, and should be unambiguous in their representation of the users. Similarly, Lee (2001) argued that the goals of visual cues should be to “*augment*” conversations, providing added detail or context to the interaction.

*Summation:*

Current 2D systems’ real estate problems can be improved with a scrollable chat space, which can have relocating views, similar to the exploration and navigation systems found in conventional 3D systems. This allows users more room to move and assemble in more natural conversation groups.

Current problems with 3D systems can be said to stem from the choice of literal (human-like/real world) metaphors for avatar and virtual environments. By using more abstract avatar representations, the need for maintenance behaviours and other associated functions would be removed, allowing a user to focus on the task of chatting.

From Spiegel’s (2001) observations, the disassociation between user and representation in two- and three-dimensional chats can be said to occur due to the inability of these systems to allow direct and continuous user control and/or influence over their avatars. The effect can be reduced by increasing the influence user conversational actions have over the user’s representations.

## 4.0. Literature Review

---

A review of current research and application of abstract visual chat systems was carried out to help develop visual communication in gridWorld. This review identified the verbal and visual communication cues and metaphors used in each application to achieve each system's goals. Through this exercise any outstanding cues, metaphors, and strategies that were proven effective can be documented towards a theoretical framework upon which future development decisions can be made.

### 4.1. Existing related studies and projects:

Current studies/research into social visualisation and interfaces for conversations online are being conducted at Massachusetts Institute of Technology's Media Lab research group: the "Social Media Group", headed by Judith Donath. Their research concerns "society and identity in the networked world" (<http://smg.media.mit.edu/>). Current research projects of interest include: Tele-Directing, Visual-Who, Chat-Circles, and Coterie. These projects focus on developing abstract interfaces for online communication systems.

#### *Collaborative Tele-Directing:*

The Tele-Direction interface allows multiple users (the Directors) to collaboratively control a shared remote resource or agent (the Actor), which can be a person, a robot or a program. The interface is designed to deliver a single instruction to the Actor through mediation among the Directors. Users collectively suggest and vote on comments and questions presented by the supervisor using a voting economy built into the chat interface. All directions voted on cause real actions by the actors in a remote location, shown in real-time as the stage in the user-Director's interface.

The user is represented by a simple text-geometry embodiment, distinguishable from each other by colour and text-name. Goals are posted by moving an avatar onto the stage and typing an action the user would like the remote actor to perform, producing a transparent circle and action-text in the poster's colour. User-Directors vote by moving onto the circle of choice within a set time limit. By setting goals directly on stage, postings and voting can be expressed visually in context with the current scene.

Chatting can take place outside the stage, where utterances are briefly displayed next to a user's avatar.

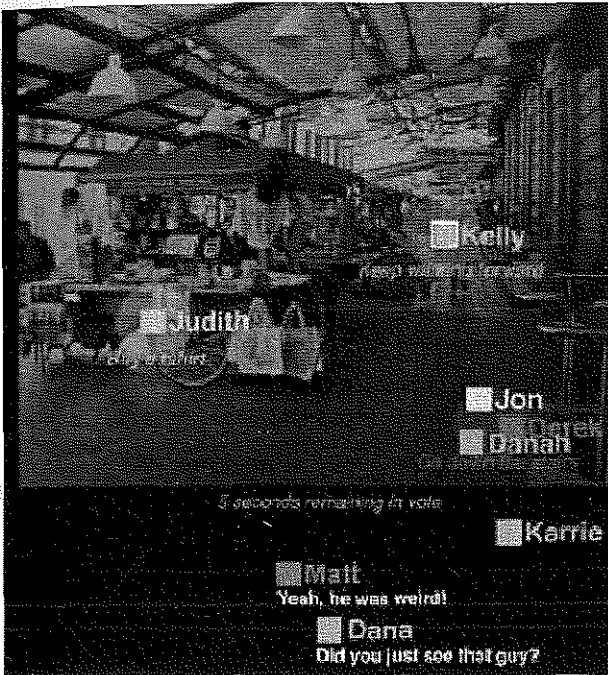


Figure 6 - The Tele-Direction interface

Tele-Director's interface is highly contextual, task driven and intuitive. The voting system implements user embodiment issues of presence, location, identity and efficiency, identified by Benford et al. (c. 1995), in context with the directing scenario. The spatial metaphor of the 'stage' provides a visual landmark where typing and expressing visual cues are intuitively and cognitively linked, as prescribed by Mania and

Chalmers.

#### *Visual-Who:*

Visual-Who was developed for intuitively visualizing large data sets of user postings. It depicts social patterns normally observable in a community, such as the formation of people into groups of peers. Visual-Who provides a means for members of an online community "to explore and understand the roles, ideas and histories that bring them together" (<http://smg.media.mit.edu/projects/VisualWho>).

Visual-Who visualizes a community's shifting patterns by measuring a user's "attractiveness" to a set of topic anchors in the virtual space. A user's attractiveness is based on the relevance of each individual's data/characteristics/interests to the community's anchors points. A user's name, which is first generated in the centre of the space, is 'pulled' towards anchors that are relevant to their preset interests. The strength of the pull is directly proportional to the relevance or interest toward a given topic. Colour is used to distinguish different 'types' of users, i.e. academics are yellow, other staffs are blue, and students are red. The community's shifting social patterns is visualized through examination of the motion/shifting of different user

names towards various anchors. The degree of a user's *presence* is expressed by the brightness of his/her name; the brightest names show the strongest posting activity and thus the strongest presence in the online community. Conversely, the darkest is seen as having been idle the longest.

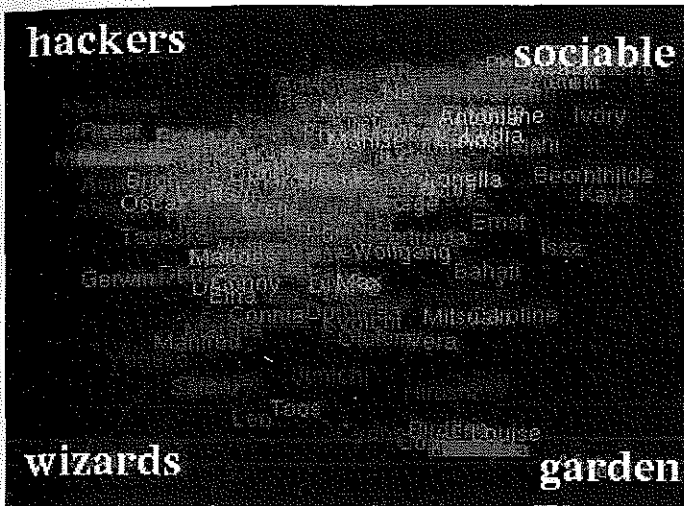


Figure 7 - Visual Who

Visual-Who is highly efficient in computer resource use, managing screen real estate, and expressing (the required) visual cues. The choices of embodiment, environment and visualization metaphors reflect and exploit the text nature of their medium to create a “synthetic geography” that acts a means

of coordinating and mapping the virtual space (Miller et al., c. 2000). Quantifiable data, such as number of people and size of community groups can be easily visualized from a quick glance at the shape, size, and movements of these ‘text-landscapes’, while qualitative information about each user in their groups is easily gathered through textual characteristics such as transparency, hue and colour.

#### *Chat-circles:*

Chat Circles is a 2-dimensional “abstract graphical interface for synchronous conversation”, created by Fernanda Viegas (1999). The user’s identity and activity is represented as a small coloured circle, which expands to accommodate the chat messages inside, much like a ‘speech bubble’. The circle grows and brightens with each message, while fading and diminishing in periods of silence.

The chat space consists of a single room for all conversations, rather than multiple separate rooms, or channels. A “hearing range” metaphor is adopted that intuitively breaks large groups into conversational clusters, where users need to be physically close to other participants to be able to “hear” (read) their conversation. Circles

outside the “hearing range” appear empty, but are still visible so as to indicate presence and activity of others.

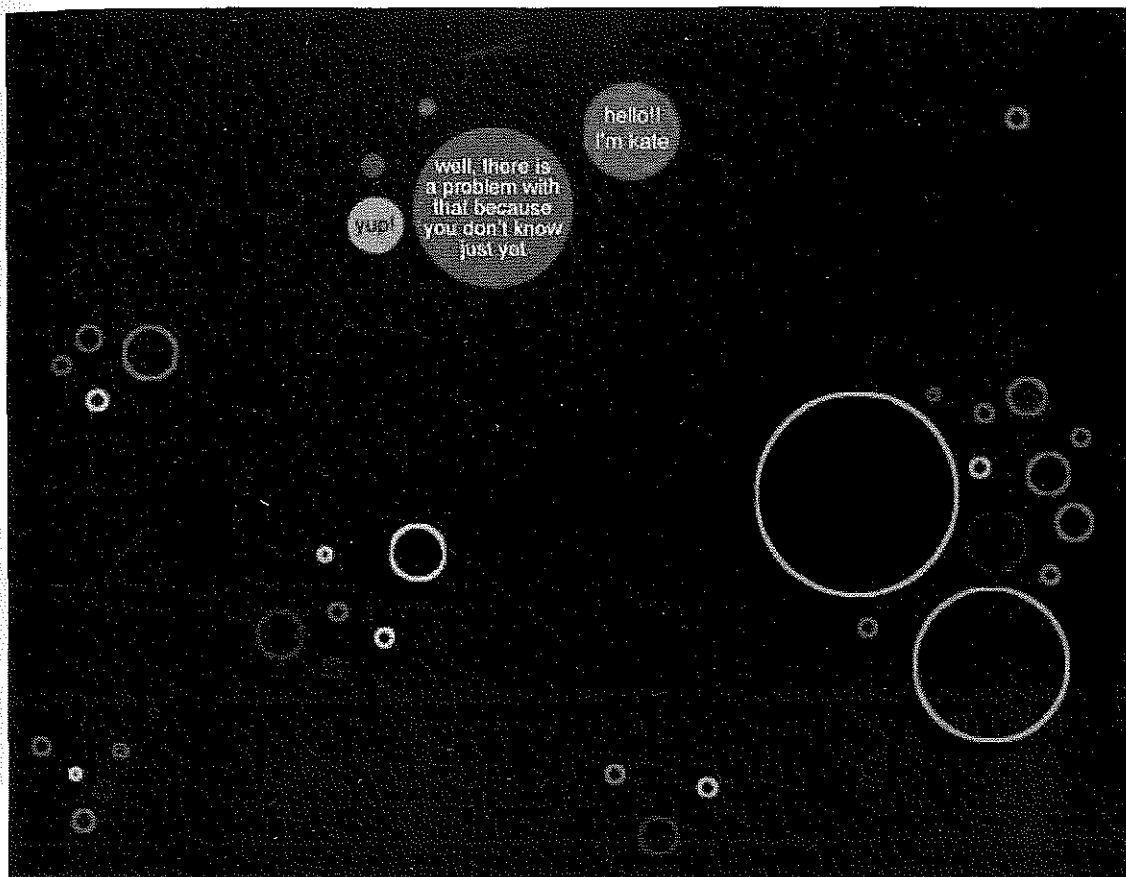


Figure 8 - Chat Circle conversation space

Additionally, an archival interface is available, providing a temporal visualization of conversation histories. A coloured vertical thread represents each user, the y-axis representing time, the intersecting horizontal bars representing moments of dialogue.

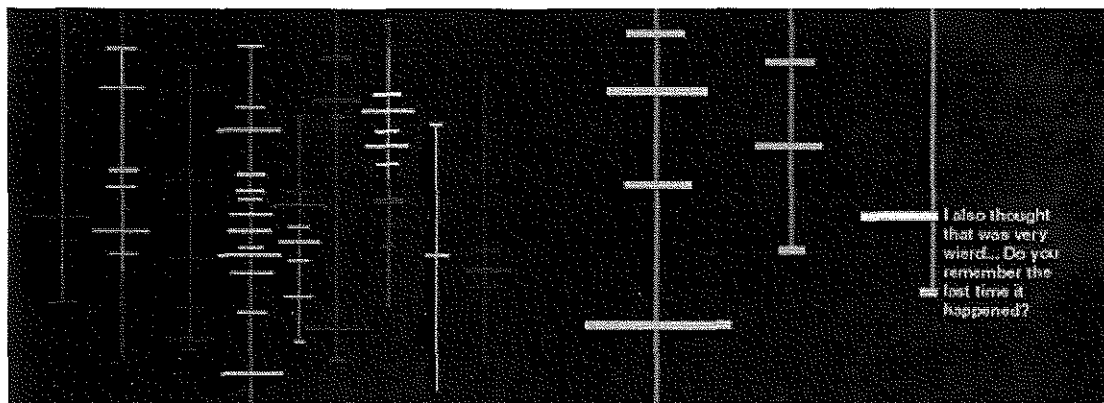


Figure 9 - Chat Circles, Archival Interface



Chat Circles provides an innovative application of an abstract graphical chat interface. It successfully addresses all user embodiment issues outlined by Benford et al. (with the exception of gesture and facial expression) with minimal system and cognitive requirements. Its conversational and archival metaphors provide clear and obvious visualization of conversations that are well suited to the “sense and substance” of the interface (Tufle 1983 as cited by Donath), namely the process of chatting.

*Coterie:*

Coterie is a 2D abstract avatar system for “creating and displaying nonverbal social back-channels in online environments” (Spiegel, 2001) in IRC channels. Visual cues are created from monitoring the social interaction with a channel over time, building statistical models based on message post rate, conversation length and memberships. Information is automatically visualized so that the passive social expressions of the individual and the group can be seen at a glance.

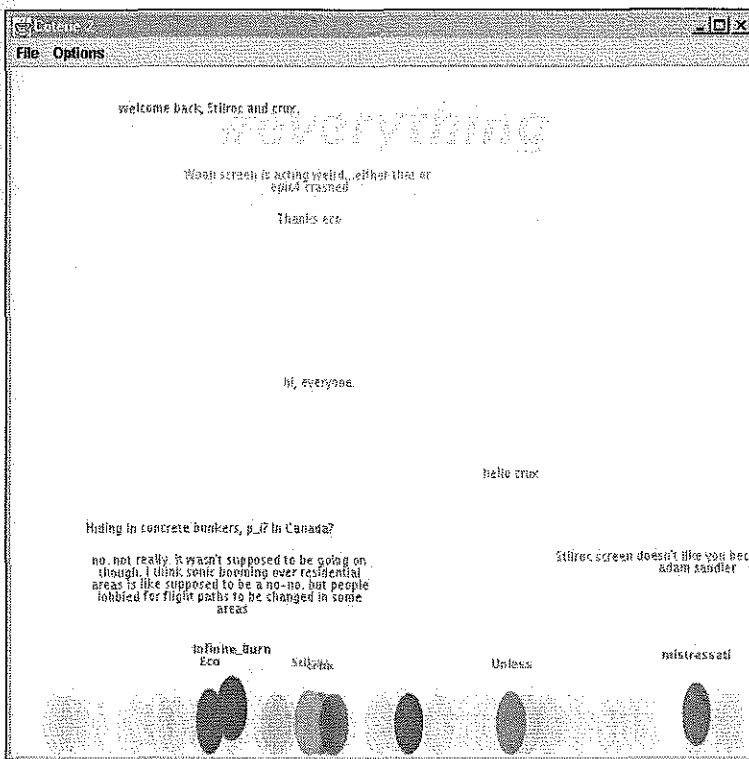


Figure 10 – Coterie

Participants are abstractly represented as ovals of various colours. The ovals are arranged horizontally based on their activity levels; participants are move to the centre of the screen when they post a message, and slowly move toward either side as they become more inactive. Users fade, change hue and transparency to depict historical social interaction information, as seen in figure 8, providing a clear visual distinction between users who are actively chatting, and those who are passively observing. The interface focuses on cohesiveness and temporal visualisation of conversation threads.

Coterie provides another example of an abstract graphical chat interface. It addresses and makes practical use embodiment issues in its visual interface. While gestures and emotional expressions are not represented, Coterie successfully conveys the visual (background) cues of presence (individual and group), activity, location, and participation/involvement using simple abstract user embodiments.

#### *4.1.1. Summary*

This review has provided some insightful strategies and proof of the effectiveness of abstract metaphors to successfully facilitate communication and interaction online.

Visual cues that (currently) are most relevant to the task of chatting: Presence, Location, Identity, Activity, Availability and Involvement, can be effectively expressed through innovative use of text and simple geometric-shapes. Tele-Directing, Visual-Who, Chat Circles and Coterie's nonverbal channels can be seen to be based around the representation's visual attributes of size, colour, hue/transparency/lightness, movement, and position. Through combining these four basic cues more abstract and deeper nonverbal messages can be communicated, such as attention, interests, or individual and community group behaviours. Further research into nonverbal communication may reveal additional non-verbal cues and their application/implications in communication.

In the reviewed systems, avatars are secondary to the chat, acting as placeholders for users in the online space. Visual cues are centred on status and condition of users. It can be assumed that social behaviours and expressions of emotions and actions are communicated through the text-channel by creative use of available ASCII characters, such as seen in traditional text-based chats. By letting emotional or behavioural content of the message be conveyed through the utterances, rather than the avatars, control and production of conversation is potentially more efficient and effective.

## 4.2. The Communication Channel

Communication is a process. It involves a source to initiate it, a message to communicate, a method to convey it and a receiver to interpret it. The process of asking a question in person is completely different over the telephone, through a letter, or in an email. While the source (speaker), message and receiver may remain the same, the means by which the message is transported is vastly changed in each situation and would affect how well the message is transferred, interpreted, and eventually replied.

A communication channel is the sum of the medium (the matter/energy unit by which transfer is made), the message (information that is carried by the medium) and the connection or link that is formed between the participants (Blake & Haroldsen, 1975).

“Communication channels are the effective links inter-connecting the source-receiver nodes in a communication structure, through which messages flow. Channels couple the source and the receiver, enabling them to communicate”  
(Rao, 1972, p.4: as quoted by Black & Haroldsen)

Berelson and Steiner (1964, p.527: as quoted by Black & Haroldsen) define communication as the act or process of transmission (of information, ideas, emotions, etc). As communication channels can be seen as the means of this transmission, the effectiveness of the communication channel used would thus determine the effectiveness of transmission, which is communication itself. It can be seen that these channels can determine credibility, feedback, involvement, availability, permanency and other dimensions of interaction, as they affect how messages would be encoded when sent, and decoded when received.

Communication is a multi-channel phenomenon (Druckman, Rozelle, & Baxter, 1982). This is most evident in face-to-face interactions, which can utilize the medium of speech, sound, sight, touch, taste and smell as in any one instance. Taylor et al. (1986) identify these as message carriers, the “things a person actually says and does while communicating” (p.9), substituting when words are inadequate or inappropriate.

Though a majority of a face-to-face interaction is carried out through a spoken or written medium, other unspoken transmissions are present and play an active part in providing meaningful communication. From observation those channels which use verbalized message carriers (e.g. spoken or written language) can be said to form the verbal channels of the interaction, those that use non-verbalized message carriers can be said to form the non-verbal channels. For example, a person shrugging would be said to be communicating through a nonverbal channel using action as the message carrier. On the receiving end, the receiver would be connected to the sender by visually witnessing the message carrier (i.e. the shrugging action).

As discussed previously, many of the nonverbal channels are almost nonexistent in online text-based interactions, as text is a verbalized message carrier. It can be seen that this deficiency has led to the inclusion of graphical interfaces to many chat systems today, with the aim of providing a medium for nonverbal cues to be expressed in some way.

#### *4.2.1 Classifying non-verbal communication*

Communication as a process requires a communication channel to connect each participant and provide a medium or message carrier to transfer the message itself. Nonverbal communication has been identified as “the transfer of meaning involving the absence of symbolic sound or sound representations” (Blake & Haroldsen, 1975, p.43), utilizing such senses as vision, hearing, taste, smell, touch and other abstract senses.

In their book “*A Taxonomy of Concepts in Communication*” (1975), Blake and Haroldsen quote Duncan (1969) in classifying nonverbal communication as the following:

- **Body motion/Kinesic behaviour** – gestures, expressions and other body movements
- **Paralanguage** – vocal qualities and speech characteristics
- **Proxemics** – perception and use of space
- **Olfaction** – smell
- **Skin sensitivity** – touch and temperature

- **Use of artefacts** – dress, cosmetics or objects

Druckman, Roselle and Baxter's "*Nonverbal Communication: Survey, Theory, and Research*" (1982) identifies the four channels of nonverbal communication as vocal (paralanguage), facial, body (kinesics) and visual (eye behaviour). In "*Communicating*" (1986), Taylor et al. identified these characteristics as message carriers in nonverbal communication, noting similar and additional message carriers of actions, appearance, use of objects, space, voice qualities, vocalizations, environmental sounds, touch, time, smell and environment, categorising them as either visual, audio/aural, tactile, olfaction (smell), or a combination of carriers. Figure 11 shows a hierarchical representation of nonverbal communication channels using the above classifications.

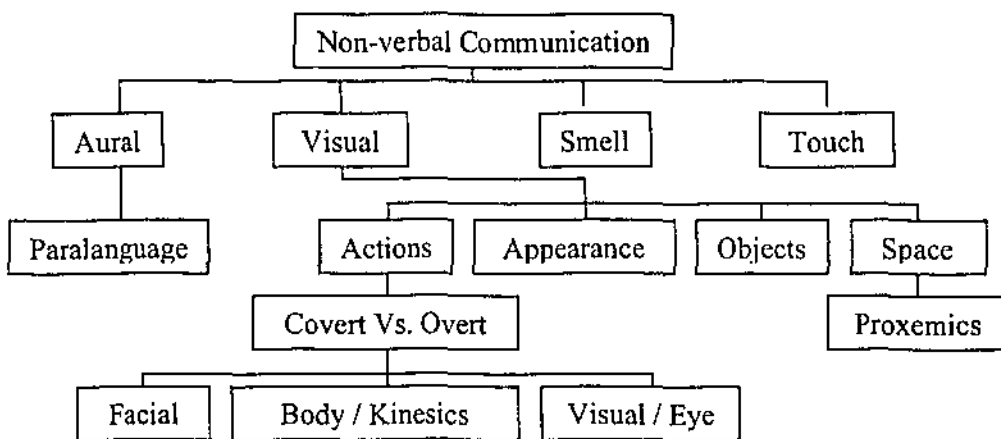


Figure 11 – Non-verbal Hierarchy

#### *Aural nonverbal communication*

Aural nonverbal communication involves nonverbal cues that are heard. This can include quality and characteristics of sound, such as loudness, pitch or tone. In the context of face-to-face communication, these attributes represent the basis of paralanguage; aural nonverbal cues that are expressed by **how** a person speaks.

Paralanguage is one of four communication channels identified by Druckman et al. (1982) in their survey of nonverbal communication behaviours. Druckman et al. note the importance of paralanguage in detecting the emotional states of conversational participants. Conversational behaviours can be regulated through intonation,

paralinguistic drawl or a drop in pitch, providing cues for turn taking. Additionally, nonverbal cues in paralanguage can extend to what is not spoken, where silence or hesitation in speech can indicate information processing.

#### *Visual nonverbal communication*

Taylor et al. (1986) identified four distinct categories in visually received nonverbal message carriers as Actions, Appearance, Objects, and Space.

Actions include the other three of the four communication channels identified by Druckman et al. in their surveys. These include Facial expressions, bodily movements or kinesics, and visual/eye behaviours such as gaze. Nonverbal cues in this category may be overt (out in the open and readily available) or covert in nature. While covert actions are harder to distinguish, they often communicate more, being harder to control and thus will send very “honest messages” (Taylor et al., 1986, p. 95).

Appearance includes the visible physical characteristics of the conversant. These are “bodily communicators that do not involve movements” (Taylor et al., 1986, p.102), such as physique, size, arrangement of body/stance and colour. These characteristics can be differentiated into stable characteristics and unstable characteristics. Stable characteristics are those such as gender or build, which suffer little fluctuations or change over short periods of time, unlike unstable characteristics, such as clothes, cosmetics or hair length, that can be more easily change at will.

Objects include any item that may be used to express nonverbal messages. Objects such as a table centrepiece or muddy shoes can influence the environment and affect how people communicate. Additionally, objects may carry symbolic meaning that can be directly associated to its owner, indicating rank, history, personality, etc.

Space, or Proxemics, involves the physical distance between conversants and/or their surrounding environments. Linked to location, space can be an important indicator of the intimacy of a conversation where the more private an interchange is, the less distance there is between participants. Taylor et al. identified interpersonal distance to be one of the more important aspects in direct face-to-face interactions (p. 96), differentiating three levels of interaction distances: social consultation/far, personal-

intermediate, and intimate-close as categorised by Halls (1966) and Store & Morden (1976). Taylor et al. note the relationship between interpersonal distances and territoriality of personal space, in which each individual possesses a 'bubble' that "expands and contracts depending on the circumstances...situations and our relationships with the people with whom we're talking" (Taylor et al., 1986, p. 103). This space can be "contaminated" by other people through changes in interpersonal distances, and thus evoking reaction and other behaviours from the space holder. Depending on the holder's relationship to the intruder, this encroachment may be met with acceptance to tension or hostility.

#### *4.2.2 Summary*

From the brief taxonomy of nonverbal message carriers identified, the nonverbal channels of paralanguage, actions, appearance, objects, and space/proxemics can be said to be most transferable and suitable in an online setting. Firstly, due to the dissociative nature of the computer interface, only those cues that can be received visually or audio/aurally can be considered as viable communication channels. Secondly, these channels are seen to be the most utilized channels in face-to-face communication (Blake & Haroldsen, 1975; Druckman et al., 1982; Taylor et al., 1986) and can be combined to express various other abstract messages (such as time). Finally, these channels are the prime carriers of the visual attributes of size, colour, hue/ transparency/lightness, movement, and position that was identified earlier as visual cues for nonverbal meaning in successful abstract avatar systems.

## 5.0 The design and implementation of gridWorld:

The GridWorld prototype is a 3d avatar-chat system attempts to facilitate communication and interaction between users online. GridWorld allows users to interact by manipulating abstract avatars from a third person perspective. All avatar movements and interactions are held on a 3d “chat floor”, and are controlled by the user using a point and click interface. Chat messages are typed in the left hand text-field. Emoticons are added into the text-messages using radio buttons above the text-field. Avatar menu controls are displayed in the black area below the chat floor window. All with utterances appear as “speech bubbles” over the speaker’s avatar for a period of five seconds and then moved to a history window below avatar menus. All users within the chat world are shown in the top-left user list. User instructions are available in a scroll box at the bottom-left corner of the screen.

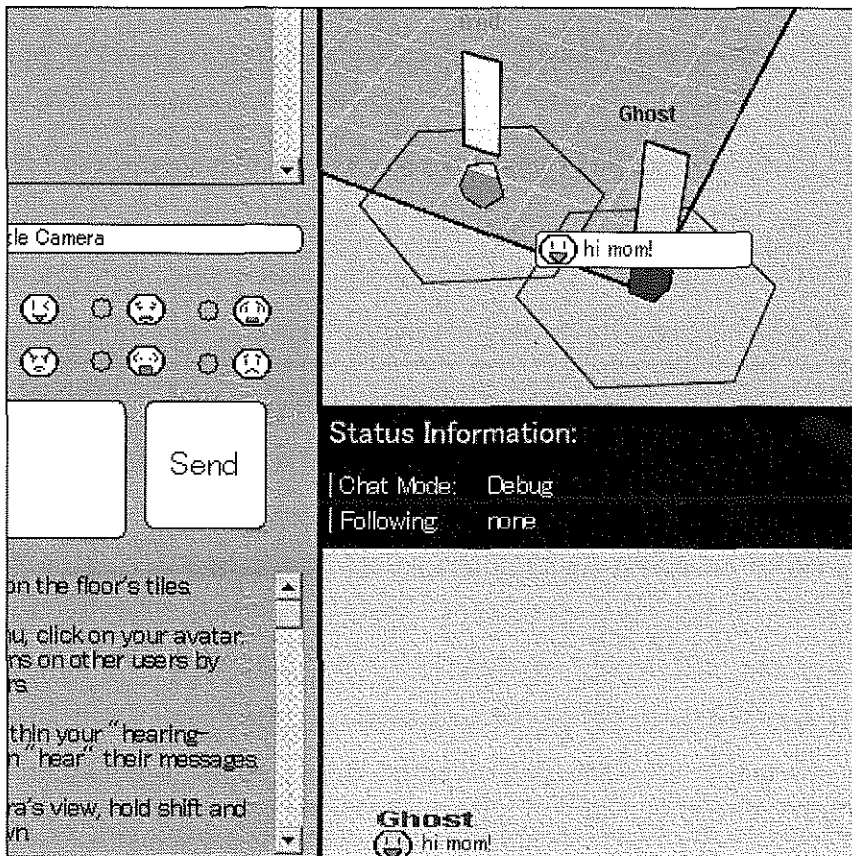


Figure 12 - GridWorld Screen Capture

The goal of GridWorld’s visual metaphors is to complement and enhance the communication process between two or more users through the visualisation of nonverbal cues of Presence, Location, Identity, Activity, Availability, and Emotional



expression using the nonverbal communication channels of paralanguage, actions, appearance, objects, and space/proxemics.

### 5.1 Designing User Representation:

The basic and most obvious feature of any avatar-based chat system are its avatars. Online avatars are representations of real-world users in a virtual environment and provide a means of communicating visual nonverbal cues normally expressed by our bodies. Through avatars, users may interact with virtual objects, travel virtual spaces and, most importantly, communicate with other (avatars) users. It can thus be seen that the capabilities and limitations of what users can “do” in any particular chat system is often dictated by the design of that system’s avatar mechanics.

#### *Abstract Avatars*

In three-dimensional graphical chats the use of highly anthropomorphised virtual bodies and behaviours can be distracting to the chatting process and often creates behavioural expectations that cannot or are difficult to satisfy (Benford et al., c. 1995; Donath, 1997; Lee, 2001; Mania & Chalmers, 1998; Miller et al., c. 2000; Vilhjálmsón, 1996). Additionally, the stilted and often uncommunicative movements of these types of three-dimensional representations offers little pretence for “believing in a close connection between the user and the user representation”, resulting in the avatars that are little more than “fancy icons” (Spiegel, 2001, p.49)

Research at MIT’s Sociable Media Group’s research into new methods of facilitating online communication provides excellent examples of the effectiveness of abstract avatars in online chat scenarios (Lee, 2001; Spiegel, 2000, 2001; Viegas & Donath, c. 1999). Chat Circles and Coterie demonstrates the ability of simple geometric shapes in the representation of an interacting user by establishing a direct and continuous connection between the user and the representation by allowing the avatar’s physical/visual appearance to be directly influenced the user’s conversational and virtual actions. The metaphors used are simple and accurate in their reflection of real world interactions. The more active or participating a user is, the more prominent their representation will appear. Similarly, the more talkative or eloquent the user is, the larger their avatar will become. Finally, conversational protocols promote the

formation of observable conversational groups, serving not only to relieve visual congestion by filtering/organising user utterances to localised nodes, but also creating discernable sociable group behaviours and attributes (such as activeness or association).

### *Embodied Conversational Agents*

In their discussion of embodiment in collaborative systems, Benford et al. identified the following issues and techniques to be considered in designing user representation: Presence, Location, Identity, Activity, Availability, Gesture/Facial expression, history of activity, user perspective, user representation through multiple media channels (text, graphic, audio), efficiency and truthfulness. Benford et al. suggest that designs should take into context the tasks and goals of the system and how these embodiment issues will affect the overall collaborative outcome, quoting: "User embodiment concerns the provision of users with *appropriate* body images so as to represent them to others (and also to themselves) in collaborative situations" (c. 1995). Thus it can be said that in graphical chat systems, the appropriateness of an avatar would be measured by its ability to visualise the creation and maintenance of conversation.

This "appropriateness" of the virtual body is also discussed in Cassell, Bickmore, Campbell, Vilhjálmsón, & Yan's *Conversation as a System Framework: Designing Embodied Conversational Agents* (1999), where avatars in graphical chat systems are identified as Embodied Conversational Agents (ECA). Cassell et al. define ECA as user representations that are "specifically conversational in their behaviours (geared towards facilitating user interaction by having) the same properties as humans in face-to-face conversations". This includes:

- The ability to recognize and respond to verbal and nonverbal input;
- The ability to generate verbal and nonverbal output;
- The ability to deal with conversational functions such as turn taking, feedback and repair mechanisms; and
- The ability to give signals that indicate the state of the conversation, as well as to contribute to new propositions to the discourse.

(Cassell et al., 1999, p. 1)

Though ECAs refer to the development of human-like representations, the behavioural concepts identified by Cassell et al. provides a useful behavioural goals for graphical interfaces to strive toward.

From Cassell et al.'s definitions, we can see how those embodiment issues identified by Benford et al can affect the ability's of the ECA. For example, for an ECA to generate verbal and nonverbal output that is meaningful to any conversational situations, it must first be 'seen' by other users, thus addressing the issues of presence, location and user perspective (i.e. camera angle). Secondly, the verbal or nonverbal output should be directed towards a specific individual or group, and thus addressing the issues of identity and availability. Finally, the verbal and nonverbal output generated should draw on the issues of user activity, gesture or facial expressions, and representation across multiple media (via text or graphics). Further, these verbal and nonverbal outputs would be influenced by the efficiency and truthfulness of these representations, affecting how easily and how seriously other receive them. Additionally, the display time of these outputs would be affected by the method a chat system chooses to implement a history of the verbal and/or nonverbal actions.

## 5.2 User representation in gridWorld

The goal of avatars in gridWorld is to perform the role conversational agents in the online world such that participants can see a direct and continuous connection between user and representation.

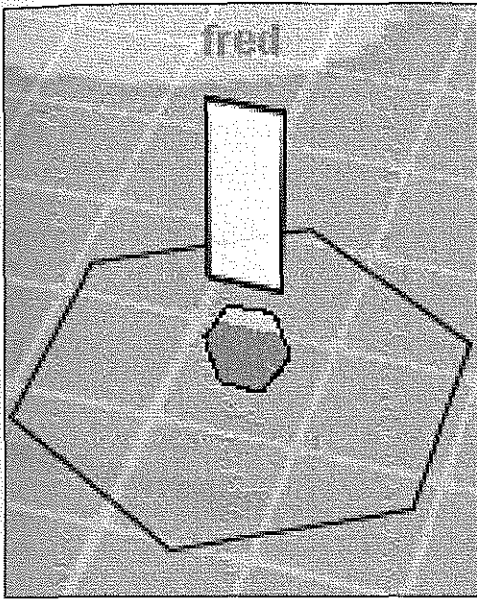


Figure 13 – An avatar in gridWorld.

The goal of the gridWorld chat environment's visual metaphor is to evoke the abstracted impression of a person looking down from a balcony onto a conference room floor. The viewer sees the interacting crowds from a top-down angle, and is therefore able to see a majority of each person's appearance, gestures and expressions. They are able to discern large groups of public gatherings from smaller, more private conversations, as well as see individuals wandering around the floors.

Using this metaphor, each individual in gridWorld is represented by a coloured sphere, giving an abstracted impression of watching a person from above. Each user's sphere's hue is customisable by the user as a means of differentiating themselves from others. Additionally, each sphere has a floating nametag as an additional means of identification, enabling a viewer to more easily associate a user's representation to a user.

Above each user is a "flag" which acts as an additional identifying marker, as well as a visual aid for expressing nonverbal cues. A user's flag shape is customisable when they enter the chat system giving the avatar a distinct appearance among other avatars. During the chat, the flag animates and changes size, hue and transparency in response to their user's conversational activities, thereby further differentiating the user from others and informing the viewer of the owner's actions and status.

Around each user is a hexagonal border, representing the boundaries of the user's hearing range. A user's hearing range must intersect with another's hearing range in

order to “hear” their utterances. This not only filters out unnecessary dialogue from other conversations, but also encourages individuals to form conversational groups much like in real-world interactions.

An essential characteristic of gridWorld’s chat perception is the visual consistency of the environment and users. In chat systems such as mIRC, messages and user names appear and disappear, or change instantaneously. Spiegel (2001) noted that “these discontinuities can lead to a distancing of the connection between users and their representations on screen” (p. 51). As such, the majority of changes to an avatar’s appearance and motion in gridWorld are smooth and gradual, allowing the viewer to see these changes occur over time. A user’s flag grows, shrinks, and fades in and out gradually, while the avatar’s movements are smooth and continuous.

While not as expressive as Comic Chat, gridWorld’s focus is on maintaining a believable connection between user and avatar. In the current version of gridWorld, avatars are static and rigid. Ideally, avatars should be able to mimic language-like deformations, such as ‘squash and stretch’ in Coterie.

### 5.3 Building gridWorld

GridWorld was build using Macromedia Director 8.5’s 3d lingo engine, and exported into Shockwave 3d format for distribution on the web. Director 8.5 was seen as the most suitable development environment for this project as it was the first development suite to fully integrate a true 3d-engine into Director Shockwave Studio, allowing for rapid design and development three dimensional, net-ready content.

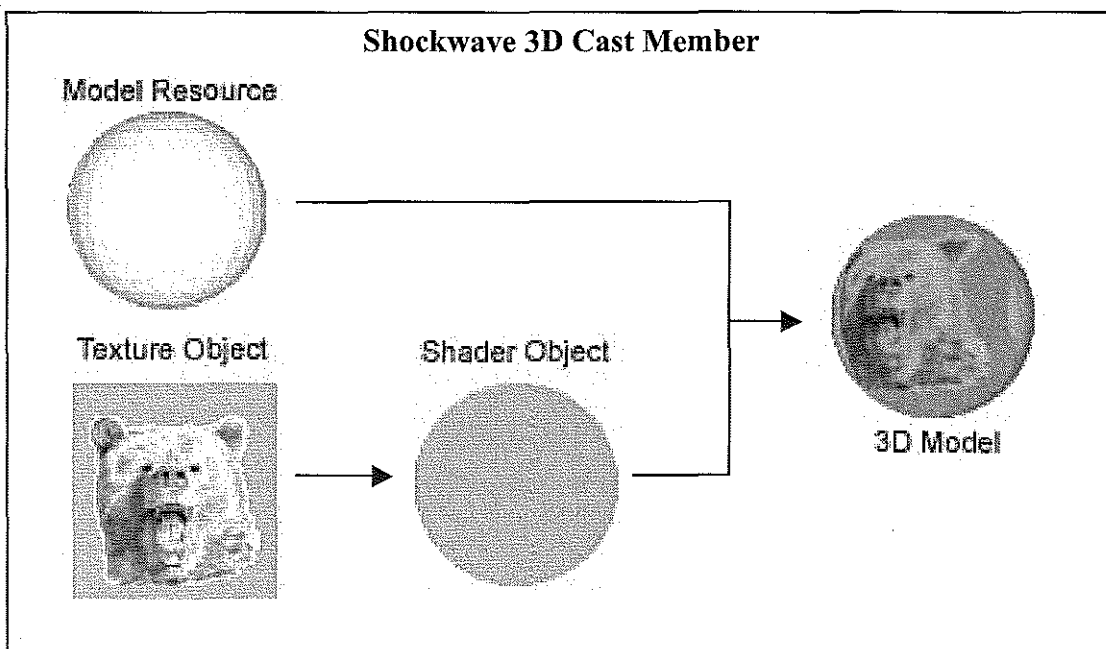
#### *Shockwave 3d and 3d lingo*

All 3d content in Director 8.5 is stored and displayed in Shockwave 3d sprites. These sprites contain within them the entire 3d world that is viewed from a camera pane. Shockwave 3d can be resized, animated, manipulated and integrated into the overall interface like any other sprites in director. The only limitation is that they require a 3d-graphics card and direct draw to render properly on screen. In this way, Shockwave 3d sprites are similar to QuickTime movie sprites, in that they cannot be overlayed or have other sprites overlayed over them.

As the Shockwave 3d sprite is itself the entire 3d world, it can thus be considered a secondary stage for 3d content. As normal sprites and functionality on the primary stage is controlled and created with Director's scripting language lingo, all functionality and 3d objects are controlled within the Shockwave 3d 'stage' through Director's new 3d lingo scripts. 3d lingo has a separate set of syntax that allows developers to directly build a 3d movie from scratch or import pre-made 3d objects into the Shockwave sprite.

### *3d object hierarchy*

3d models in Director 8.5 are created using model resources and given surface textures by assigning Shaders to their faces. Model resources are the blue prints of each 3d model, specifying its shape, size, vertices, faces, and other variables. Typically, model resources can be one of five primitive types: plane, box, sphere, cylinder, and particles. Alternatively, a user can define a customised mesh object that may be used as a model resource. Once a model resource has been created and has all relevant variables defined, it can be used to create a model in the 3d world. To add texture to a 3d model, a user must first define a Shader object to be attached to the model's surface faces. This shader object defines how each face is to be rendered when the model is drawn. Attributes like shininess, reflectivity, and colour can be set in a model's shader object. Additionally, texture objects can be attached to a shader. The following diagram illustrates the hierarchy of 3d objects in Director 8.5.



### *Floor, lights and camera*

The current gridWorld consists of a single 700 x 1000 unit 3d plane that is divided into a 50 x 50 unit grid. The plane is easily customisable in length, width and grid size, and acts as the floor of the chat environment. The ratio of the floor's length and width to grid size is vital when designing and/or customising gridWorld. The floor must be evenly divisible by 50 units in order to form uniform tiling of the floor's surface. These tiles are the foundation of gridWorld's point-and-click interface, serving as coordinates for avatar movement and placement during the chat, as well as a referencing system for rendering the floor's texture (this will be covered later).

Lighting is provided by an ambient and two directional lights positioned at 45 degrees to each other. This provides lighting for the top, top-front and top-back of the 3d world. The camera is positioned at a vector(-250, 551, 500), so as to give users an angled overhead view of the chat environment. The camera is set with a field of view of 45 degrees, giving a natural sense of depth to the floor. These values were derived from trial and error to optimise user field of vision, orientation and aesthetics.

### *Avatar modelling*

User avatars are made of three 3d primitives – a box, a sphere and a cone. The sphere primitive represents the body of the avatar. The sphere's model resource has a low resolution so as to reduce the number of polygons rendered, as well as give the avatar some physical features to accentuate its rolling movement. The box and cone primitives function as the avatar's clothes, being attached to the sphere object in a parent-child hierarchy. The box primitive was modelled with only front and back faces and with very little width so as to appear as a two-sided flag. Its width's scale could later be increased and/or decreased, allowing the box's front and back faces to move further away or closer together. The cone was the most complex of the three. Its model resource was modified to have a large top radius, a small bottom radius, and short height with only its side faces visible. This created a shallow bowl shape that sits underneath the avatar's body to form the hearing boundary.

To create a cell-shaded effect, a toon shader was used to surface the avatar and an inker modifier was added to each primitive to provide bolded edges. The toon shader applied a two-tone cell shading effect on the sphere that masked its low resolution, but

retained the sharp corners of the avatar. The inker modifier allowed the edges of the flag and hearing boundary to be seen when completely transparent.

### *Texturing*

For bitmaps to be used as textures in Shockwave 3d their dimensions need to be in the power of two (i.e. 2, 4, 8, 16, 32, 64, 128). The avatar's flag object used a 128 x 256 pixel, 32-bit depth bitmap with an 8-bit alpha channel. This same texture could be reused with different flag shapes so long as the flag's length to height ratio was the same as that of the texture.

In order to allow for a scalable grid floor size, the floor bitmap texture needed to be generated on the fly each time the 3d world is remade. Using imaging lingo, gridWorld can draw a rectangular bitmap of any dimensions to serve as the floor's texture. GridWorld uses a tiling algorithm based on a 64 x 64 pixel bitmap tile for each grid square on the chat floor. The floor texture is redrawn each time the 3d world is recreated, based on the specified length, width and tile size of the floor plane.

### *Camera Overlay*

GridWorld uses the camera's overlay function to display user nametags and message posts. Overlay objects are textures attached to a camera's overlay layers and displayed in front of all other objects in the Shockwave 3d sprite. A camera can have multiple overlay layers, each holding a single texture object. An overlay layer with a higher layer number will be drawn over all lower layers, much like sprites in the score window. Each overlay layer can be moved around the visible camera screen by altering its horizontal and vertical values in the same way a sprite can be translated with locH and LocV on the stage. Additionally, each layer's transparency and scale can be manipulated at will.

GridWorld assigns two dedicated overlay channels for each user; the first for the user nametag, the second for message posts. The nametag remains constantly visible, and is automatically repositioned directly above each avatar after each frame. The message layer remains invisible until a user posts an utterance, upon which the message texture is drawn (using imaging lingo) and (re) attached to the message overlay layer. As with the nametag layer, the message layer is also repositioned after



every frame. As an additional feature, each message layer will calculate if it is intersecting with any lower level message layers and will adjust its vertical position to avoid covering other user's posts in each redrawing.

#### *Multi-user Server*

GridWorld uses chatML to support its back end chat functions. ChatML is a multi-user server XML protocol being developed by Jacky Chong to support dynamic data and media transfer within the shockwave environment.

GridWorld uses chatML functions to transfer message postings, avatar behaviours and statistics, coordinates and other information to animate and synchronise all active gridWorld chat applications.

## 6.0. Multi-channelled nonverbal communication in gridWorld

---

Communication is a multi-channel phenomenon. Much like real-world interactions, gridWorld uses multiple communication channels to express nonverbal cues. Though restricted to those audio and visual carriers, avatars in gridWorld are still able to communicate through use of paralanguage, actions (movement, gestures and expressions), appearance, objects, and space.

### 6.1. Paralanguage

Paralanguage has been called “the implicit aspects of speech” (Hehrabian, 1972, as quoted by Druckman et al., 1982), playing a crucial role in communicating emotional state or content of an utterance, advertising to the listener cues for coordinating social exchanges, as well as influencing other’s perceptions of the speaker. Though more commonly associated with spoken language, paralanguage can also be found in the written medium. Punctuation marks such as exclamations marks, quotation marks or question marks are used frequently to add or change the tone and meaning in sentences. The use of capital, bold or italicised text, or stylised fonts creates undertones and adds volume to words and phrases.

Text-based Paralanguage
...No...
No!
No?
<b>NO</b>
<i>No</i>

Figure 14 - Text based paralanguage use punctuation and formatting of written text to provide the "vocal qualities" of speech.

In chat systems such as MSN Messenger, graphical emoticons can be freely combined with written text, supplying additional emotional context to user utterances. Similarly, emotive paralanguage is simulated in gridWorld by a combination of emoticons and text. Users can select one of nine emoticon’s to augment their speech.

Paralanguage plays a large role in gridWorld’s chat interface. As gridWorld uses an abstract chat interface, avatars do not have faces or bodies to express the users emotional states, nor any means of gesturing. This removes the need to divert attention or resources towards extraneous maintenance behaviours (Lee, 2001). Rather, the interface relies on






Emotive Paralanguage	
	No
	No
	No
	No
	No

Figure 15 - The inclusion of emoticon graphic adds additional emotional context to the text.

creative text-based and emotive paralanguage to communicate any underlying emotional or meaningful content within each utterance. This provides a much more honest representation of face-to-face interactions, where to properly understand an individual's emotional expressions there must be a mutual understanding of the conversation's situation, context, focus and intent (Vilhjálmsson, 1997).

By relying on paralanguage as the main carrier of expressive nonverbal conversational cues, the majority of communication can take place through the text-channel. This results in a quick, economical and efficient chat interface.

## 6.2 Objects

Objects in real world interactions can provide an auxiliary means for carrying nonverbal information that may be as effectively conveyed through other channels due to their medium's physical or practical limitations. An architect can more effectively describe a building design with the aid of a model. In gridWorld, the avatar's design limits its expressive capabilities to simple geometric or iconic-based actions such as blinking, rolling or deformations. However, any actions or changes applied to the avatar must be subtle enough to ensure that its original appearance remains intact and/or recognisable, and does not conflict with its abstract nature by evoking behaviour expectations. Thus, for example, the avatar cannot suddenly sprout arms to wave.

To compensate its limited functionality and avoid maintenance behaviours, each avatar has a flag and a hearing boundary object with which to communicate a majority of overt and covert nonverbal behaviours. The flag object provides cues to aid in identity and recognition, emphasises/enhances user visibility (and hence presence), and can animate and change in visual characteristics to express nonverbal communication through actions and appearance. For example, as a user's average message rate and length increases, the avatar's flag grows larger and more solid to indicate a stronger user presence. The hearing boundary object circles the avatar to

provide a visual indicator for more covert cues relating to space and proximity. Avatars must be in range of these boundaries to be heard. These two objects provide the avatar with additional channels of nonverbal expression that was previously unachievable by itself.

### 6.3. Actions

All avatar actions in gridWorld are performed automatically in response to the user's chatting activity. As a result, the representations are directly and continuously linked to its user without the need for any manual behavioural controls. This simplifies user control and creation of avatar behaviours, but does not take away any authenticity of their representation of the user's activities, such as is done by random gestures and avatar movements in Active Worlds.

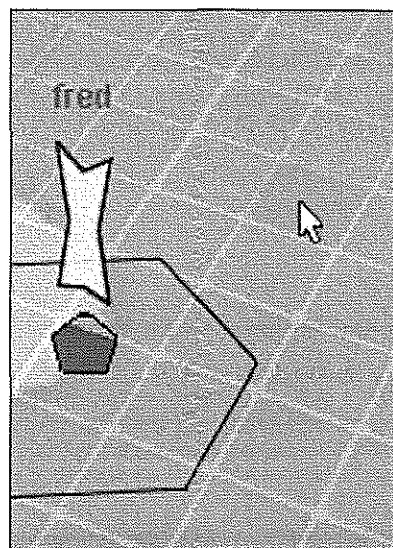
#### *Walking around gridWorld*

A user controls their avatar's movements by a point-and-click interface. Movement is based on a grid-tiled system (hence the name gridWorld), allowing only one avatar to stand on one grid-tile at any one moment. Users need only click on an available tile to direct their avatar's movements on the floor.

When a user clicks on a tile, the avatar "walks" to the targeted location. When walking, an avatar rolls along the floor, while its flag waddles from side to side, giving an observer an abstract impression of someone walking across the floor. Through this motion, an avatar is given a believable movement action that works towards the visual consistency of the chat environment.

#### *Thinking response feedback*

An avatar's flag will expand and rotate while displaying a "..." texture whenever a user begins to type in the chat field. If the user stops typing, the flag will continue to



**Figure 16** – The mouse pointer will highlight the tile to move to.

animate for a period of five seconds, then return to its normal state. Depending on the length of the message being typed, the flag will expand to several times its normal width. Short utterances will result in a small expansion, while longer utterances will result in wider expansions.

The flag metaphor models the use of silence and hesitation in real world face-to-face interactions, where:

“hesitations, unfilled pauses, and reduced verbal productivity may signal processing and decoding of speech...(enabling) the encoder to process thought into proper words and gestural forms to be spoken ” (Bruneau, 1973, as quoted by Druckman et al., 1982, p. 49).

These pauses are more easily recognized as preludes to replies in face-to-face situations, as the encoder's body language would be clearly visible to the onlooker. Additionally, these pauses in offline situations tend to be brief when compared with pauses between replies in online conversations. Pauses in online situations would be lengthened by the time taken to type a response, and if too prolonged, may be interpreted as un-received or ignored by the receiver. This problem is further compounded if the receiver is a very slow typist.

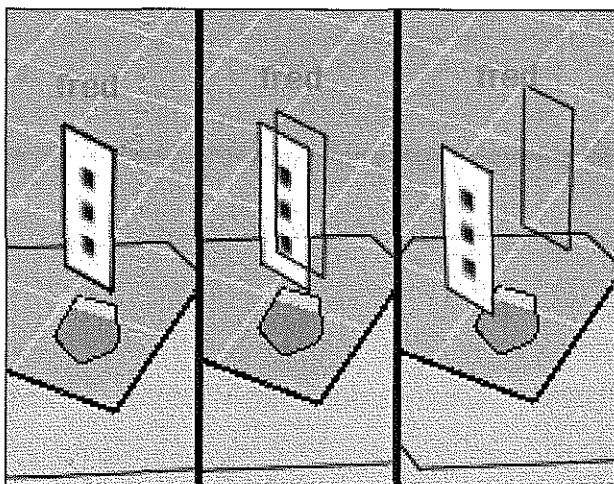


Figure 17 - An avatar's flag will expand and rotate when the user types.

By providing visual confirmation of a user typing a response in real time, both observers and conversantes have confirmation of when a user is typing, as well as an indication of the length of response. This provides a 'buffer-zone' between interchanges, satisfying the expectant sender of an upcoming reply and informing other conversantes of the

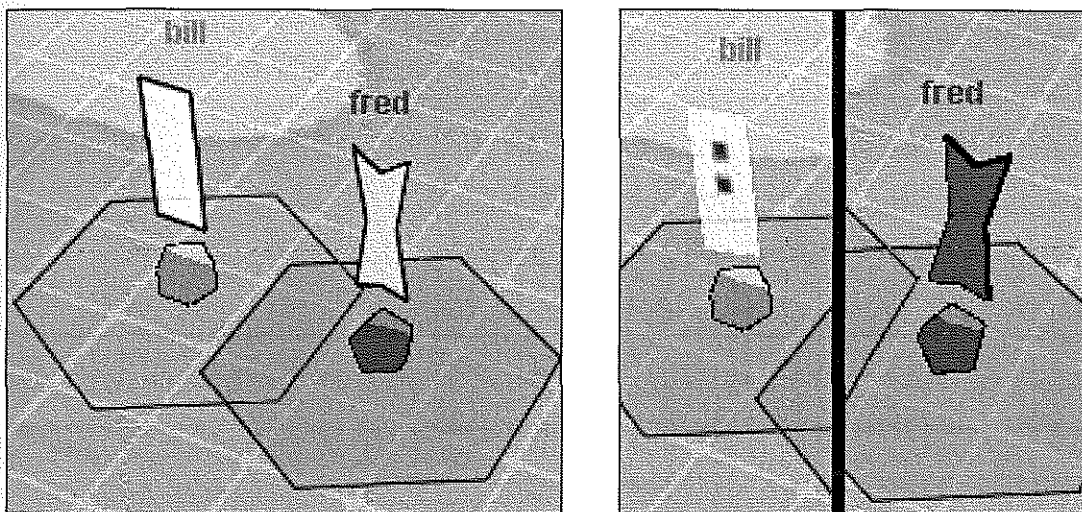
encoder's intent and activeness, allowing for regulating of conversational activities such as turn taking.

## 6.4. Appearance

### *Colour and Shape*

Colour and shape serve as means of distinguishing and differentiating one user from another. Much like clothes and cosmetics in the real world, gridWorld allows users to choose their avatar's colour and shape, giving users some measure of self-expression in their appearances. Hue is good referent for a person's identity as it is a good carrier of discrete, unchanging information (Spiegel, 2001, p. 55). Combined with geometric shape a more striking contrast between avatars can be created to further aid in identifying users. Users can select one of seven hues (orange, red, green, blue, yellow, purple and aqua) and four flag shapes, allowing for 112 unique avatar appearances.

As well as a visual cue for individual identity, colour is also used as a means of expressing chat group association and chat classes.



**Figure 18 - public, private, and host class avatars**

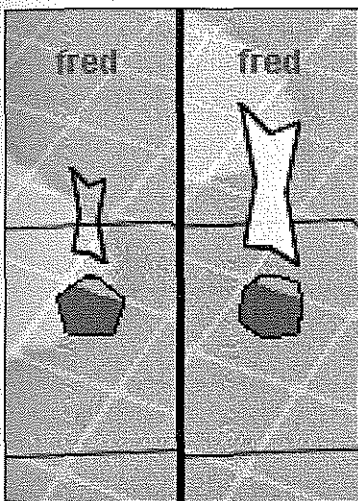
Private chat groups are distinguished by the colouration transparency of their hearing ranges. When a user creates or joins a private chat, their hearing ranges changes colour to match the host avatar's hue. Thus in a private chat, all participants would have a similar visual trait, signifying their association to the chat group as well as distinguishing them from other private and public groups.

Each avatar in gridWorld can belong to either public, private and host class. The importance of the user's ability to identify these classes is particularly important when trying to join private chats. To join a chat, a user must first click on a host avatar to

request entry into the group. If a user is unable to find the host, then they cannot join. GridWorld uses the colour and transparency of an avatar's flag as message carriers of class. An example of this is in figure 18, where 'bill' is a private avatar and 'fred' is a host avatar.

### *Size and transparency*

Size and transparency are used as indicators of the user's level of activity within the chat. Much in the same way that people can have more presence in real world conversations; a user's chat presence becomes more prominent as he/she posts more messages. As a user's chatting activities increases, their avatar's flag grows larger and more solid – thus becoming more visually dominant to a viewer. Conversely, if a user stays quiet over a period of time, their flags begin to shrink and fade, becoming less noticeable to the observer.



**Figure 19** - An avatar's flag will shrink and become transparent after a long period of inactivity.

The size of a user's flag is a measure of the average length of their utterances. GridWorld measures and records the number of characters in each utterance and calculates the average message length with periods of one minute. Based on this measure, a user's flag will grow or shrink after every minute.

Similarly, the transparency of an avatar's flag indicates how talkative a user is, based on the number of messages posted per minute. If a user post less than two messages within a minute, their flag will fade, while if a user post more than one message within a minute, their flag will grow more solid. As with changes in size, changes in transparency will also in/decrement after every passing minute, allowing for a gradual transition in appearance.

The combined effect of size and transparency creates a natural contrast between active and inactive users. As changes to these visual characteristics are gradual and directly linked to each individual's conversational behaviours, the nonverbal data expressed are truthful and accurate. The overall effect created adds an additional dimension to

the conference-room metaphor, where not only will the observer be able to see and identify individuals and groups, but will be able to also see the how active the conversations are, who is the focus of attention and who are the bystanders.

## 6.5. Space and location

Physical space plays an important role in face-to-face interactions (Blake & Haroldsen, 1975; Druckman et al., 1982; Taylor et al., 1986). This also holds true in graphical chats, as iconic, graphical, 3d or abstract avatars require a 'physical' virtual space on the screen in which to be seen to interact. In graphical chat systems such as The Palace, avatars could be placed anywhere on the screen and be able to chat to anyone. This type of implementation allows users to hold conversations without moving closer to each other, allowing interaction from opposite ends of the 'room'. This can be seen to be an ineffective use of screen space as well as interpersonal space, leading to visual clutter of disorganised avatars and their messages. This problem was avoided in Chat Circles through use of a 'hearing-range' that filters out message post that are at a distance to the user avatar. The hearing range mimicks real-world interactions by encouraging users to move closer to each other to talk to each other, thus forming clusters of user conversational groups.

GridWorld incorporates a similar hearing range metaphor, using the hearing boundary object to mark the edges of the user's personal bubble. In order to 'hear' or be 'heard' their avatar's hearing boundary must be in contact with each other. User utterances outside this contact will appear transparent and faded, much like background noise in a busy room that is heard but can be easily ignored.

The concept of territoriality has been extended in gridWorld through incorporation of private chat groups. In Chat Circles, hearing ranges act as territorial spaces for mimicking interpersonal distances of individuals, thus leading to the formation of conversational groups. Through this action, screen space is taken up and becomes 'owned' by users. In the same way a user can hold private territory, groups of people can hold a collective territory that is formed from a coalition of related interactions. This territory increases and the number of members increases, and thus consumes more space as it grows. When users create or join private chat groups in gridWorld,



their hearing boundary solidifies and takes on the hue of the host to form a hearing space. In this way, the chat group's territory is visualised by means of the collective area of their hearing spaces.

### *6.6. Summary*

GridWorld's chat metaphor allows for nonverbal communication through use of paralanguage, actions, appearance, objects and space in a virtual setting. Combining emoticon graphics with words in user messages creates a form of paralanguage that allows users to be more expressive with their utterances. Avatar flag and boundary objects give the avatar additional methods communicating actions, appearance, objects and space. Through use of the flag object, avatar's can better communicate actions (such as walking and thinking) for conversational feedback, as well as colour, shape, size and transparency for identity and status. Through use of the hearing boundary object, additional nonverbal social and conversational cues involving space can be better expressed.

## 7.0. Post development / User feedback

---

### 7.1. QUIS – Questionnaire for User Interface Satisfaction

GridWorld was tested using a modified Questionnaire for User Interface Satisfaction (QUIS) developed by Shneiderman in 1997 (Shneiderman, 1998) to evaluate the subjective effectiveness of an interactive computer system's Object-Action Interface model. The Object-Action model is based on the performance of direct manipulations applied to visual representations of objects and actions in a user interface. QUIS evaluates the effectiveness of a system's metaphors for visualising user actions and system objects by measuring the user satisfaction with the system after use.

The QUIS questionnaire is based on a series of psychometric rating scales and comment boxes arranged into hierarchical sub-components in a set of system and performance categories. These categories include:

1. User Experience,
2. Overall user reactions,
3. Learning the system,
4. Online conferencing (chatting),
5. System information/feedback,
6. Screen layout, and
7. Visual (graphic and text) quality.

A sample of the modified QUIS form used for evaluating this project can be found in appendix 1.

### 7.2. Results of QUIS Questionnaire and user feedback

User feedback was categorised into 7 parts; User Experience, Overall User Reactions, Learning the system, Chatting, System Feedback, Screen Layout, and Visual Quality. Each category is composed of several subsections of system and performance questions and their associated rating scales. At the end of user interface evaluation, the scores are tallied into averages and mapped onto 5 charts showing gridWorld's interface performance in each category. Those questions that were answered "N/A" were not included in the results calculation. Each score is measured along a scale from

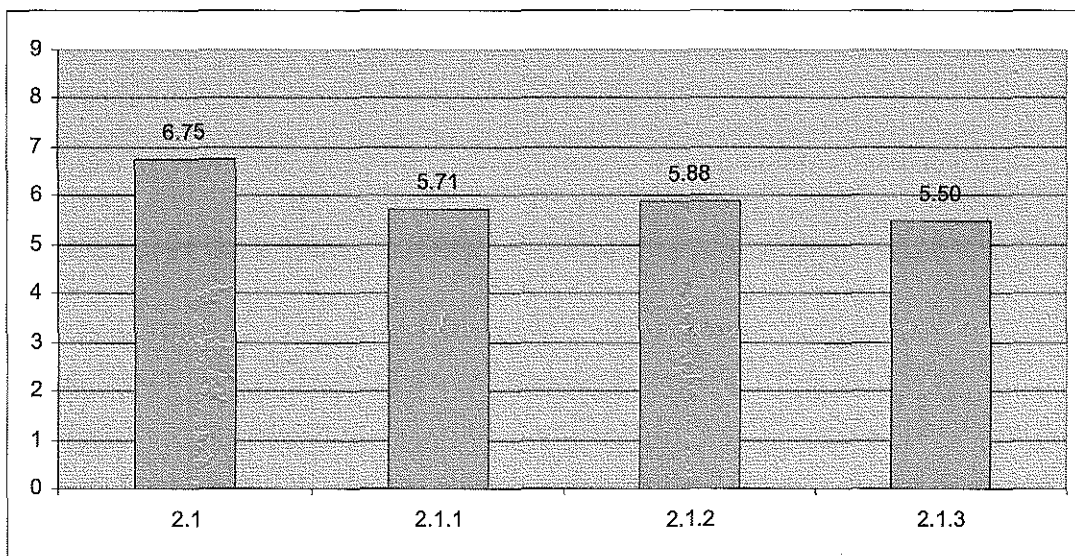
1 to 9, where 1 point indicates the strongest negative response and 9 points indicates the strongest positive response. Any score that ranked above 5 was considered to be a positive feedback. A score of 0 was given to results that were not applicable to the scale ranking measure.

### *Results of Part 1: User Experience*

All users who completed the QUIS questionnaire were experienced computer users who have previously used MSN Messenger, mIRC, ICQ and other chat programs. Most users were familiar with chatting online using text-based chat system, but were unfamiliar with graphical chats with the exception of one who had previously used The Palace. It was thus expected that the one user who had previous experience with graphical chat systems would have less difficulty learning and using gridWorld, but be more critical in their responses to the system.

### *Results of Part 2: Overall User Reactions*

The result of the user evaluation shows a positive response in overall user reactions to the system. Overall user reactions (2.1) scored 6.75 points on the QUIS ranking scale, indicating that users were adequately satisfied with the overall performance of gridWorld's interface.



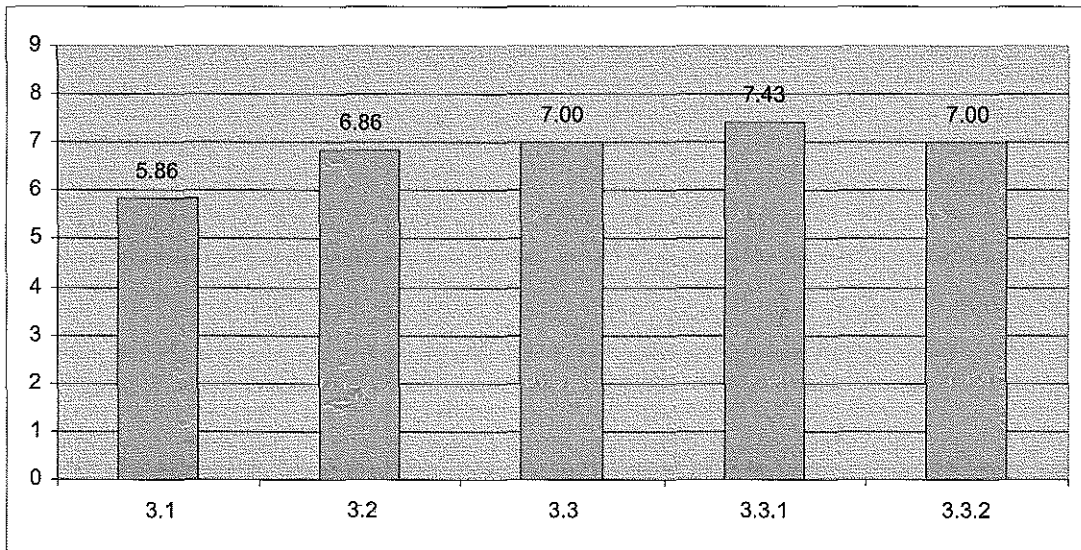
**Figure 20 - Overall User Reactions**

While the overall feedback was positive, sub-questions evaluating the underlying factors for this satisfaction (satisfaction found chatting on the system, level of stimulation of chat metaphor, and ease of use) showed weak positive scores hovering

just above 5 points (a neutral score), indicating that users only found chatting, chat metaphor and system intuitiveness to be only moderately satisfying. This shows that, while the current interface is usable, it can still be vastly improved on in future iterations.

*Results of Part 3: Learning the system*

Results from part three of the questionnaire shows the current system to be only moderately easy to learn, scoring 5.8 points for ease of learning to operate the system (3.1). Question 3.3, “task can be performed in a straight-forward manner” and its corresponding two sub-categories (3.3.1 and 3.3.2), scored an above average score of 7 and higher, indicating that users were generally satisfied with the interfaces task-logic sequence and operations while learning to use the system. Question 3.2, “exploration by trial and error”, scored below the 7-point average as seen in 3.3, indicating that the current interface did not offer enough encouragement for exploration.



**Figure 21 - Learning the system**

When we compare scores in each questionnaire, question 3.2 almost always received a lower score than 3.3. Thus it can be assumed that the low user satisfaction in learning the system is largely caused by the current interface’s tendency to intimidate almost half the users away from trial-and-error exploration of the system’s capabilities.

#### Part 4: Chatting

Results from evaluation of the chatting category provided both encouraging and discouraging results. Overall scores in this category showed relatively low satisfaction in the interface's ability to support chatting online. Results showed that users found holding conversations (4.1) to be only relatively easy. While the number of people allowable in a conversational group (4.1.1) was shown to be satisfactory, the users expressed only a slight satisfaction to the ease of establishing private conversations (4.1.2) within the chat world. Question 4.2 evaluated the appropriateness of the chat window size of 1024 x 768 with a YES/NO answer (thus showing a score of 0 in the chart), to which all but one user answered Yes.

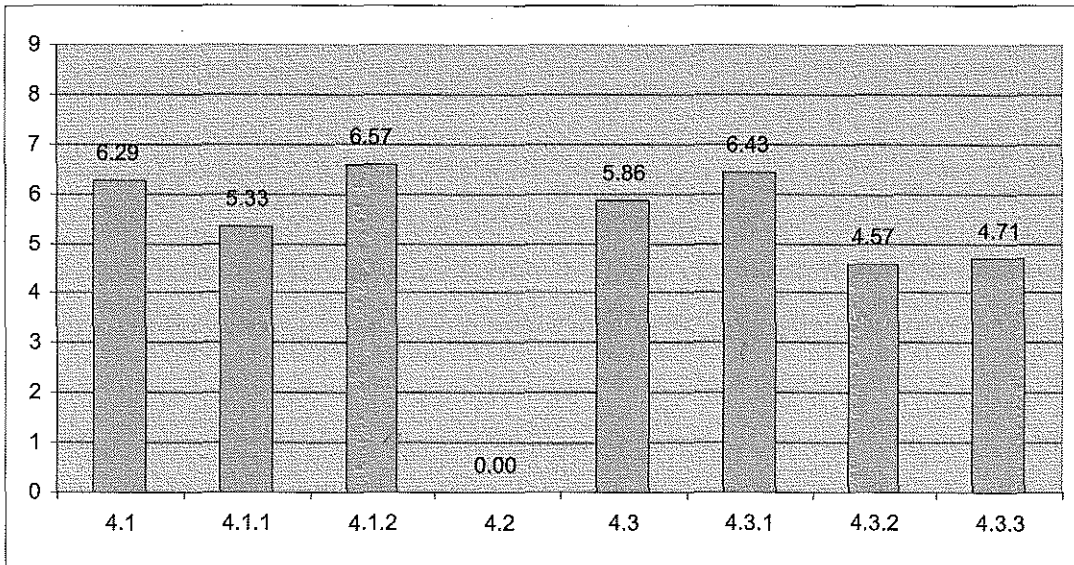


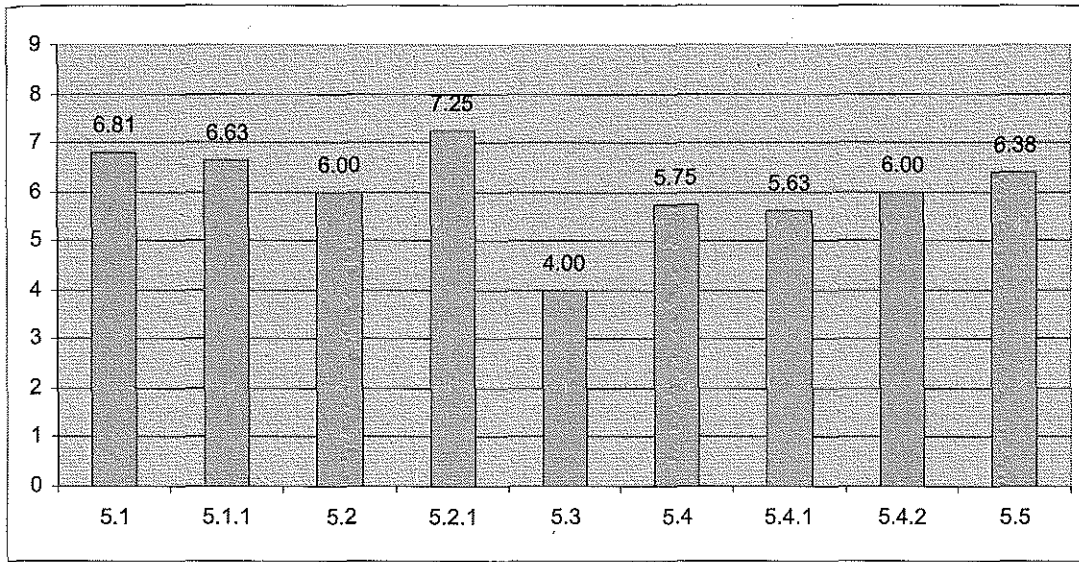
Figure 22 - Chatting

Question 4.3 and its sub-questions evaluated the bulk of the avatar's visual metaphors. Scores from question 4.3 showed that users were moderately satisfied with the system's avatar metaphors/visual cues in determining the focus of attention during online conversations. Users were relatively satisfied with the ease at which an avatar's visual cues helped them to identify an active speaker (4.3.1). However, scores for ease in which to identify a lead speaker (4.1.2) and lurkers (4.1.2) showed slight user dissatisfaction.

#### Part 5: System/Feedback

The users reports in this category indicate relative satisfaction in the clarity of system instructions that appear on screen (5.1) and the functionality/purpose of system menus, buttons and/or fields (5.1.1). While users were satisfied in the predictability of

results in menu selections (5.2.1), users showed less satisfaction in the system's ability to keep them informed of their tasks and operations (5.2). Users found the system's error messages to be moderately unhelpful (5.3).



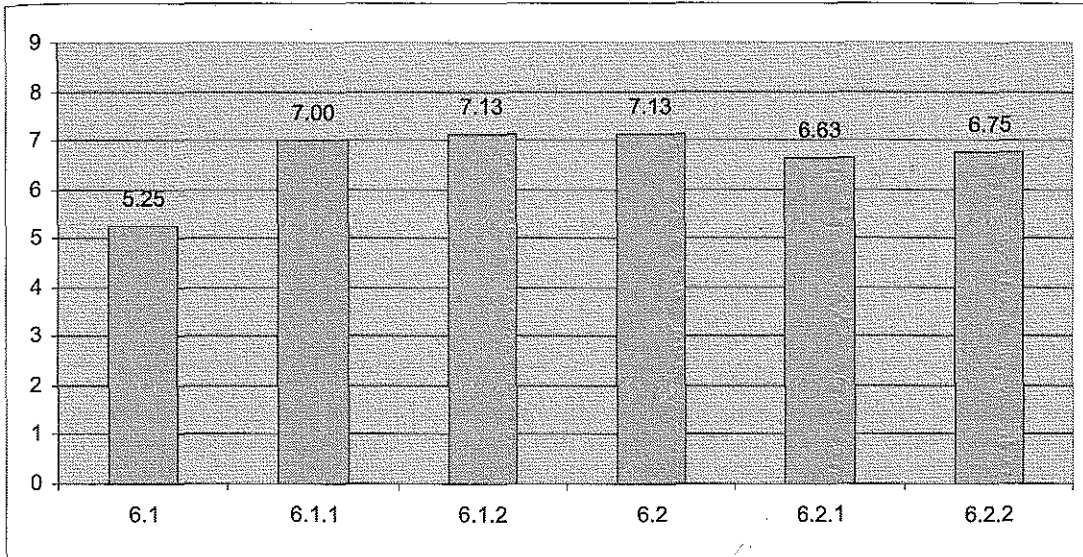
**Figure 23 - System/Feedback**

Users showed only a small satisfaction in the avatar controls (5.4), finding the response time of most avatar-related operations to be only slightly adequate (5.4.1). However, users showed a greater satisfaction with the amount of effort required to perform those avatar operations, reporting that the effort level to be relatively unnoticeable (5.4.2). Lastly, users reported their relative satisfaction of the reliability of the system (5.5).

#### *Part 6: Screen Layout*

Overall, evaluation of screen layout returned relatively high scores, indicating that users were generally satisfied with this category. While clarity of text characters (6.1.1) and font legibility (6.1.2) received high scores, average score for readability of text characters on screen (6.1) barely scored above neutral. This may be a result of text rendering problems experienced when gridWorld is run on a computer with insufficient video memory, causing text to appear incomplete in the main chat window while drawing correctly in the history window.



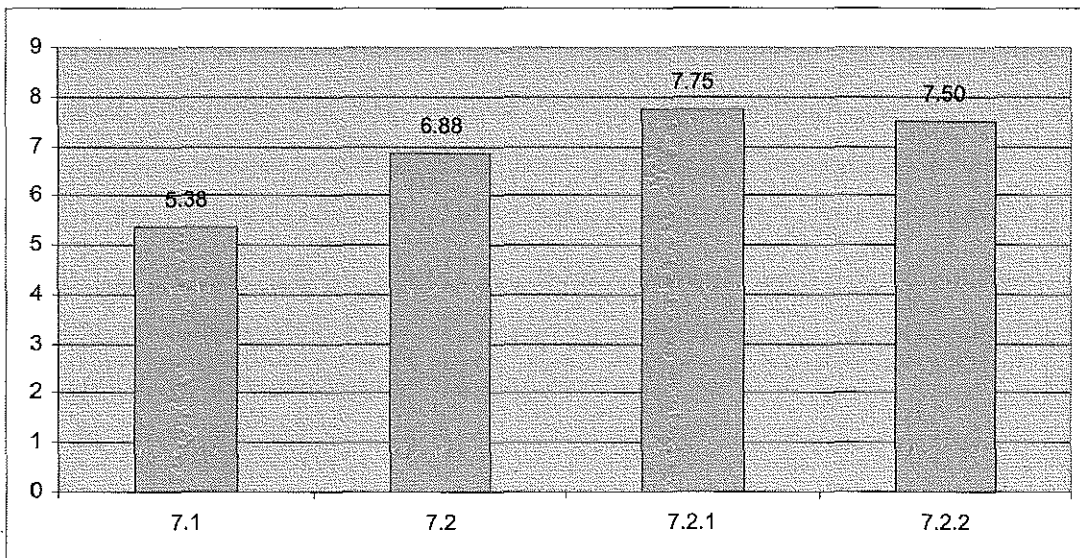


**Figure 24 - Screen Layout**

Further results indicate that users thought the screen layout was almost always intuitive (6.2), reporting that amount of information that is displayed on screen was moderately adequate and the arrangement of information displayed on screen sufficiently logical.

*Part 7: Visual Quality*

Results from this category showed that users were only slightly satisfied with the quality of emoticon graphics (7.1). The users appeared to be satisfied with implementation of the 3d world (7.2), reporting the avatar and world objects to be sufficiently clear. Users thought that the colours employed in gridWorld were natural (7.2.1) and the range of avatar colours available was adequate (7.2.2).



**Figure 25 - Visual Quality**

### 7.3. Analysis of system usability

While the users responses indicate a general satisfaction with the current interface, the relatively low positive scores show that major improvements are still needed. Analysis of the feedback received through the QUIS questionnaire and observations of users during evaluation suggest problems in the following areas.

#### *Problems with learning the system*

Low scores for chatting satisfaction and ease of using the system can be seen to be linked with the low scores found in learning the system. Currently all instructions are displayed in the scroll box on the bottom-left hand corner of the screen. User instructions consist of an itemised list of operations and features that are available in gridWorld, ranging from basic movement and chatting to private channel management. To save on space and scrolling, all instructions were brief and to the point as possible, and where at the time, thought to be adequate enough to guide users in their exploration of gridWorld's operation. The interface was assumed to be intuitive enough to enable users to learn the system through experience, and trial and error.

While this system worked satisfactorily for learning basic task and operations, such as moving and chatting, the more abstract and complex controls required more detailed explanation than the instruction list or on screen system feedback could provide. This would suggest that, while user instructions where not detailed enough, system messages and feedback were also offered insufficient explanation and/or information to be of use by themselves. Results from question 3.3.2 showed that feedback that the interface provided was not always clear, while question 5.3 showed user dissatisfaction with error messages. Without adequate information on the purpose and operation of a menu or whether an executed task or operation was a success or failure, users may not be able to predict the direct consequences of their actions. An uncommunicating system would only serve to cause anxiety to the user, and may act as a hindrance the exploration by trial and error.

Finally, post examination showed that the instruction list was also difficult to find, where one user commented that the instructions were hard to see, and so was not aware of a majority of chat operations and features available in the system.



### *Problems with controlling/tracking user activities*

The results of the evaluation show that users experienced some problems controlling avatar and chat operations in gridWorld. Low user satisfaction that was reported in the avatar controls, which was shown to become frustrating during times of high system lags (due to congestion of the multi-user network), where avatar movements become delayed. Additionally, gridWorld's point and click system was reported to be less than perfect, with some clicks on tile and avatars being ignored by the system. This problem disrupts the chatting process and accounts for the low user satisfaction with avatar controls as well as in the response times of avatar reactions.

When a user sets his avatar to follow another avatar, their following status is changed from "none" to the name of the user to be followed, which is displayed in the black menu bar below the chat world window. Post examination and user feedback has suggested that this information about user action/status is hard to see, and is usually not noticed when it changed. Secondly, when users click on floor tiles to select a position to move to, users complained of not receiving any acknowledgement of that command or any visual reference of the target path or location. This would account for the low satisfaction with the system's capabilities of keeping users informed on their current tasks and operations, and is another example of insufficient system feedback.

### *Problems with private chatting*

Observation of users during evaluation showed that users were having problems setting up private chat channels. Users experienced difficulty entering the channel name due to improper syntax and the existence of another private channel using the same name. In the first case, users failed to notice the menu instructions specifying the need for a "@" character in front of the channel name. In both cases, when the operation to create a private channel failed, a majority users could not determine the source of the problem and gave up trying, thus accounting for the low satisfaction scores received in establishing private conversations as well as the reported dissatisfaction with the helpfulness of error messages. Again, the problem is seen to be linked to insufficient system feedback, as well as inadequate explanation in the user instructions.

### *Summary of analysis of system useability*

From the previous analysis, all major problems seem to be linked with the issue of inadequate feedback, information or instructions. While some problems can be sourced to displaying issues and program bugs, the main problem with the useability of gridWorld is that users do not know how to (properly and fully) use gridWorld.

## 7.4. Analysis of effectiveness of nonverbal metaphors

User response to the chat metaphor was not as positive as expected, with users reporting gridWorld's visual metaphors to be only mildly stimulating. Examination of "Chatting" results reveals more detail on which visual cues were more readily accepted and which cues were not.

### *Effectiveness of Paralanguage*

Though low, results measuring user reactions showed that users gained some satisfaction out of chatting on gridWorld. Almost all users reported to have previously used MSN messenger and so were familiar with the use of emoticons with text as a form of paralanguage. Most users were quick to make use of this similar system used in gridWorld's chat interface, some commenting the need for gridWorld to adopt a larger range of emoticon expressions and graphics. While all user responses show that the current emoticon-text chat system could be better improved, they also show the relative success of paralanguage as the main carrier of expressive nonverbal conversational cues through the text-channel.

### *Identifying 3D Objects*

Support for user satisfaction with the use of 3d objects in gridWorld is evidenced in "Visual Quality" results. Observations of users during the evaluation showed they were able to recognise avatar objects and orientate themselves on the chat floor. While quick to identify the use of their avatar's sphere and flag objects, it took some more time and the occasional explanation to discover the purpose of the hearing range boundary.

### *Effectiveness of Avatar Actions*

Avatar waddling actions while walking were found to be effective and entertaining method of depicting user movements around the chat floor; with several users commenting how they liked the way the avatars move. Determining the focus of attention during conversations and the ability to identify when a user was speaking scored relatively high, indicating the effectiveness of the avatar's flag animations at expressing the visual cues of response and user activity. One user was particularly impressed with the correlation between an avatar's flag rotation and scale, and the length of the message being typed.

Users however, commented that the point-and-click controls of the avatar movement were sometimes clumsy and unresponsive, rating a low satisfaction with avatar controls. These lags cause delays in an avatar's reaction and become a source of visual inconsistency that distracts and disrupts the chat process.

### *Effectiveness of Avatar Appearance*

The relatively high satisfaction points scored in the evaluation of the use of and range of colour suggest that users were comfortable with the implementation of colour in distinguishing identity in user representations. Moderately high positive scores for the implementation of 3D objects also suggest user satisfaction with the use of flag shapes to further promote identity and user distinction.

While users seem satisfied with identifying each other through colour and shape, the use of size and transparency to communicate presence was largely unnoticed. Users still expressed some difficulty at identifying individuals leading conversations from those who were lurking. Only one user noted the correlation between the size and transparency of an avatar's flag and it's user active involvement within a group conversation. It can be seen that a longer chat session was needed to provide users with the opportunity to fully recognise these long-term cues, as well as more users to provide a more visible spectrum of activity levels. Given that the evaluation was carried out within a time period of 15 minutes and with a maximum of 3 users, it can be seen that the current implementation of size and transparency metaphors fail to express the presence in small user numbers and short chat durations.

### *Effectiveness of Space and location*

The categories of holding conversations and the number of people (allowable) in a conversational group scored high satisfaction points, providing additional evidence that users were comfortable with the use of hearing range in the chat metaphor. However, observation of user actions showed that users initially were not aware of the existence of a hearing range, and only noticed the relevance of the hearing boundary object after some time through trial and error, discovery of instructions or explanation, thus accounting for the relatively low user satisfaction with chatting and learning the system.

### *Summary of analysis of effectiveness of nonverbal metaphors*

Those nonverbal visual metaphors that experienced problems appear to have been unsuccessful or misunderstood due to awkward/limited implementation and inadequate explanation. The system of size and transparency to express level of user presence was broke down during conversations that lasted only short durations and involved small user numbers, while users were ignorant of the relationship between the hearing boundary and the hearing range. While the metaphor for user presence failed because of poor execution, ignorance of the purpose and use of visual cues such as the hearing range can be seen to be linked to the reoccurring issue of inadequate system information and instruction.

## 7.5. Future Improvements And Possibilities

Analysis of the QUIS questionnaire and user feedback revels several problems with chat interface usability and nonverbal metaphors that will need to be addressed in future versions of gridWorld. Along with these corrections, future versions will endeavour to improve the interface through revision of chat and system menus and GUI as well as implementing more advanced features and functionality that will add value to the system.

### *Improving interface useability: Part 1 – improved user instructions/User help*

While analysis showed that users were comfortable with the operation of gridWorld's basic functions of moving and public chatting, many users were uninformed of the operation (and sometimes existence) of more advanced chat features, such as creating

and managing private chat channels/groups. In order to increase interface useability, clearer and more helpful instructions is needed to train, as well as provide support for users as they become more familiar with the system's operations and capabilities.

Future implementation of gridWorld will need to incorporate a more comprehensive help system than the currently available instructions list. The help system will be accessible through a help button, which can open up in a new browser window or a MUI dialogue box. The help system will support basic to advanced tutorials as well as an index of operations and definitions explanations with examples.

#### *Improving interface useability: Part 2 – improved user control & tracking*

Users experienced lags in avatar movements/animations and chat message pop-ups when users generated multiple mouse clicks or messages too quickly. These lags between user action and system reaction occur when user input overloads the network and causes the multi-user server to slowdown. To minimise these lags, future versions of gridWorld will revise the current implementation MU procedures to optimise the efficiency of sending and receiving of XML data packets. For example, current procedures for system wide synchronisation of gridWorld chats is to have each chat send avatar coordinates to every user on the network every three frames. A better option would be to store and distribute all user data server side through server scripts. Users need only to send a their own coordinates (and other statistical info) and retrieve onscreen user data, eliminating the need for the MU server to send information on users that are not seen.

Secondarily, additional chat world GUI's will be implemented to help improve user's ability to track their actions. These include iconic or symbolic (animated) rollover cursors, which will inform users of the actions of mouse clicks on specific objects in the chat environment, and field markers that will highlight geographical information like target coordinates/user, movement paths and other relevant information.

#### *Improving interface useability: Part 3 – floating menus/improved system feedback*

Currently, gridWorld displays all system and avatar menus appearing outside the chat floor window. This method proves ineffective of menu display as it forces the user to divert their attention away from the chat scene to operate the menu. Additionally, the

menu display may be completely overlooked if the user is fully engrossed in chatting. While it was originally intended for this project to employ floating menus in its interface, lack of time and 3d Lingo resources prevented its implementation. However, now that the gridWorld prototype is functional, more time can be invested into developing this and other OS functionalities through use of extras such as OSControlXtra.X32 or MUI for accessing avatar, chat and other system controls.

Through use of floating menus and MUI dialogue boxes more screen space can be dedicated to the chat world window as well as providing more room for longer and more comprehensive system feedback, instructions and other chat/system relevant information, such as error messages that identify problem, description, suggestion/instruction for improvements, and examples or references.

#### *Dynamic emoticon-text messaging*

Paralanguage in gridWorld is based on attaching emoticon graphics to sentences or paragraphs to give them emotional content and intonation. Based on MSN and Yahoo chats, the gridWorld emoticon-text system would allow users to apply add emotional tone to any number of sentences in a single posting by attach various emoticons before, within or at the end of each phrase. The current chat messaging system is limited to one emoticon graphic per user utterance, thus the user can only attach only one emotion/intonation to any chat posting. Future versions of gridWorld would feature an improved emoticon-text system that would allow users to insert more than one emoticon graphic at any line position within a posted message.

#### *Improved avatar animation and visuals*

While avatar movements and animations were received satisfactorily, they are stiff and do not flow as well as intended. Avatar flag scale and transparency jumps suddenly as user statistics change during conversations, when they should flow smoothly from one state to another, depicting changes in one effortless transition. This would allow avatars to more easily express small status changes during short term and small group interchanges.

## 8.0. Conclusion

---

At the beginning of this thesis, it was identified that the main deficiency of text-chats was their capacity to express nonverbal cues (particularly covert cues) that are so readily used in face-to-face conversations. To rectify this, graphical chats incorporate emoticon/graphics, typically in the form of avatars, to provide visual nonverbal communication. The use of avatars to represent users in the online world created issues of embodiment, such as efficiency, appropriateness truthfulness, and control, particularly among the graphical 3d chat. It was seen that current 3d chats tend to favour a literal visual metaphor, trying to rebuild the real world in a virtual setting, thus giving rise to performance and control issues, such as maintenance behaviours. In these systems the mutually exclusive arrangement of chat and avatar behaviour controls meant that their avatars became little more than fancy icons, which offered no real meaningful improvement in nonverbal communication than the text-chats.

GridWorld was presented as a graphical 3d chat that offered true nonverbal communication in an online setting through use of abstract avatars and visual metaphors. Based on a theoretical framework established through research in abstract interfaces for online communication conducted by MIT's Social Media Group (<http://smg.media.mit.edu/>) as well as additional literature in nonverbal communication cues, the gridWorld interface is able to express multichanneled nonverbal behaviour that are directly influenced by the user's conversational activities.

While evaluation of gridWorld and its abstract interface revealed some deficiencies in its nonverbal metaphors and implementation, overall, gridWorld performed satisfactorily in supporting online communication through both verbal and nonverbal channels.

## Reference:

---

- Baker, J. (1990). *Online Emotional Discourse, gender language and style in computer-mediated communication*. Retrieved 9 August, 2002, from the World Wide Web: <http://sjsu.sjweb.net/1/author.html>
- Benford, S., Bowers, J., Fahlén, L. E., Greenhalgh, C., & Snowdon, D. (c. 1995). *User embodiment in collaborative virtual environments*. Retrieved 16 March, 2002, from the World Wide Web: [http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/sdb\\_bdy.htm](http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/sdb_bdy.htm)
- Blake, R. H., & Haroldsen, E. O. (1975). *A Taxonomy of Concepts in Communication*. Ontario: Hastings House, Publishers, Inc.
- Cassell, J., Bickmore, T., Campbell, L., Vilhjálmsón, H., & Yan, H. (1999). *Conversation as a System Framework: Designing Embodied Conversational Agents* [on-line PDF]. Retrieved 20 September, 2002, from the World Wide Web: [http://gn.www.media.mit.edu/groups/gn/publications/ECA\\_GNL.chapter.to\\_handout.pdf](http://gn.www.media.mit.edu/groups/gn/publications/ECA_GNL.chapter.to_handout.pdf)
- Donath, J. S. (1997). *Inhabiting the Virtual City: The design of social environments for electronic communities* [on-line]. MIT. Retrieved 20 April, 2002, from the World Wide Web: <http://persona.www.media.mit.edu/Thesis/ThesisContents.html>
- Druckman, D., Rozelle, R. M., & Baxter, J. L. (1982). *Nonverbal Communication, Survey, Theory, and Research* (Vol. 139). Beverly Hills, California: Sage Publications, Inc.
- Lee, M. W. (2001). *Chatscape: A behaviour-enhanced graphical chat built on a versatile client-server architecture*. Massachusetts Institute of Technology, Massachusetts.
- Mania, K., & Chalmers, A. (1998). *A classification for user embodiment in collaborative virtual environments* [on-line PDF]. Dept. of Computer Science, University of Bristol. Retrieved 16 March, 2002, from the World Wide Web: <http://www.cs.bris.ac.uk/Tools/Reports/Ps/1998-mania.pdf>
- Microsoft. (1996). *Microsoft Comic Chat*. Retrieved, 2002, from the World Wide Web: <http://research.microsoft.com/vwg/projectsheets/comicchat.htm>
- Miller, S., Mitchell, K., Eva, S., & Wood, J. (c. 2000). *Geospatial information visualization user interface issues* [on-line PDF]. Retrieved 1 April, 2002, from the World Wide Web: <http://www.geovista.psu.edu/sites/icavis/agenda/PDF/Cartwright.pdf>



- Shneiderman, b. (1998). *Designing the user interface, Strategies for effective human-computer interaction* (3rd ed.). USA: Addison-Wesley.
- Spiegel, D. (2000). *Creating and displaying nonverbal social back-channels in online environments*. Unpublished Thesis Proposal, Massachusetts Institute of Technology, Massachusetts.
- Spiegel, D. (2001). *Coterie: A Visualization of the Conversational Dynamics within IRC*. Massachusetts Institute of Technology, Massachusetts.
- Taylor, A., Rosengrant, T., Meyer, A., & Samples, B. T. (1986). *Communicating* (4th ed.). Englewood Cliffs, New Jersey: Prentice-Hall, Inc.
- Viegas , B. F., & Donath, J. S. (c. 1999). *Chat Circles* [on-line]. MIT Media Lab. Retrieved 6 April, 2002, from the World Wide Web:  
[http://smg.media.mit.edu/papers/Viegas/ChatCircles/chat-circles\\_CHI.html](http://smg.media.mit.edu/papers/Viegas/ChatCircles/chat-circles_CHI.html)
- Vilhjálmsón, H. H. (1996). *Avatar Interactions* [on-line]. Retrieved 1 April, 2002, from the World Wide Web:  
<http://web.media.mit.edu/~hannes/project/index.html>
- Vilhjálmsón, H. H. (1997). *Autonomous Communicative Behaviors in Avatars*. Massachusetts Institute of Technology, Massachusetts.

## Bibliography:

---

- Boyd, D., Lee, H.-Y., Ramage, D., & Donath, J. S. (2002). *Developing legible visualizations for online social spaces*. Unpublished Conference Paper, MIT, Cambridge MA.
- Damer, B. (1998). *Netiquette and Community Hosting* [on-line]. Retrieved 1 April, 2002, from the World Wide Web: <http://www.digitalspace.com/avatars/book/appendix/netiq.htm>
- ELF. *Virtual Learning Environments* [on-line]. Retrieved 1 April, 2002, from the World Wide Web: <http://www.elf.ufl.edu/virtual/vle.html>
- Evaluation Tools*. [on-line]. Retrieved 24 September, 2002, from the World Wide Web: [http://mime1.marc.gatech.edu/MM\\_Tools/evaluation.html](http://mime1.marc.gatech.edu/MM_Tools/evaluation.html)
- Kessler, G. D. (c. 1999). *Virtual Environment Models* [on-line PDF]. Lehigh University. Retrieved 16 March, 2002, from the World Wide Web: <http://vehand.engr.ucf.edu/handbook/Chapters/chapter13.PDF>
- Miller, H. (1995). *The Presentation of Self in Electronic Life: Goffman on the internet* [on-line]. Department of Social Sciences, The Nottingham Trent University. Retrieved 14 June, 2002, from the World Wide Web: <http://ess.ntu.ac.uk/miller/cyberpsych/goffman.html>
- Miller, S., Mitchell, K., Eva, S., & Wood, J. (c. 2000). *Geospatial information visualization user interface issues* [on-line PDF]. Retrieved 1 April, 2002, from the World Wide Web: <http://www.geovista.psu.edu/sites/icavis/agenda/PDF/Cartwright.pdf>
- Morley, David and Robins, & Kevin. (1995). *Culture, community and Identity. Spaces of Identity: Global media, electronic landscapes and cultural boundaries*, 43-69.
- Newman, W. M., & Lammig, M. G. (1995). *Interactive system design*. Cambridge, Britain: Addison-Wesley.
- Nonverbal Communication Theories*. [on-line]. Retrieved 24 September, 2002, from the World Wide Web: <http://www.orst.edu/instruct/comm321/gwalker/nonverbal.htm>
- QUIS: The Questionnaire for User Interface Satisfaction*. [on-line]. Human-Computer Interaction Lab. Retrieved 19 October, 2002, from the World Wide Web: <http://www.cs.umd.edu/hcil/quis/>
- Rodenstein, R., & Donath, J. S. (2002). *Talking in Circles: A spatially-grounded multimodal chat environment* [on-line]. Retrieved 14 June, 2002, from the World Wide Web: [http://www.media.mit.edu/~royrod/projects/TIC\\_CH2000/index.html](http://www.media.mit.edu/~royrod/projects/TIC_CH2000/index.html)

- Taylor, T. L. (1999). Life in Virtual Worlds: Plural existence, multimodalities, and other online research challenges. *The American Behavioural Scientist*, 43(3), 436-449.
- Viegas , B. F., & Donath, J. S. (c. 1999). *Chat Circles* [on-line]. MIT Media Lab. Retrieved 6 April, 2002, from the World Wide Web: [http://smg.media.mit.edu/papers/Viegas/ChatCircles/chat-circles\\_CHI.html](http://smg.media.mit.edu/papers/Viegas/ChatCircles/chat-circles_CHI.html)
- Vilhjálmsson, H. H. (1996). *Avatar Interactions* [on-line]. Retrieved 1 April, 2002, from the World Wide Web: <http://web.media.mit.edu/~hannes/project/index.html>
- Zabiliski, R. (2001). What are the key elements when designing a user friendly computer interface? *Your place, my place, interface*, 28-40.

## Appendix 1 – QUIS Sample Questionnaire

---

# Questionnaire for User Interface Satisfaction

## Part 1: User Experience

Please tick the length of you computer experience

- |  |   |
|--|---|
| <input type="checkbox"/> Less than a day               | <input type="checkbox"/> 6 months to less than 1 year |
| <input type="checkbox"/> 1 week to less than 1 month   | <input type="checkbox"/> over a year                  |
| <input type="checkbox"/> 1 month to less than 6 months |   |

Of the following, please tick the chat systems you have used previously.

- |  |                                     |
|--|-------------------------------------|
| <input type="checkbox"/> MSN Messenger | <input type="checkbox"/> Yahoo Chat |
| <input type="checkbox"/> Lycos         | <input type="checkbox"/> Excite     |
| <input type="checkbox"/> mIRC          | <input type="checkbox"/> The Palace |
| <input type="checkbox"/> Active Worlds | <input type="checkbox"/> Other      |

If other please specify:

## Part 2: Overall User Reactions

2.1	Overall reaction to the system	Terrible				Neutral				Wonderful					
		1	2	3	4	5	6	7	8	9	10	11	12	13	NA
2.1.1	I found chatting on this system to be	Frustrating								Satisfying					
		1	2	3	4	5	6	7	8	9	10	11	12	13	NA
2.1.2	The chat metaphor was	Dull								Stimulating					
		1	2	3	4	5	6	7	8	9	10	11	12	13	NA
2.1.3	I found using the chat system to be	Difficult								Easy					
		1	2	3	4	5	6	7	8	9	10	11	12	13	NA



**Part 5: System / Feedback**

5.1	System instructions which appear on screen	Confusing	Neutral	Clear	1	2	3	4	5	6	7	8	9	NA
5.1.1	Menu/button/field commands or functionality	Confusing		Clear	1	2	3	4	5	6	7	8	9	NA
5.2	System keeps you informed about what task/operation you doing	Never		Always	1	2	3	4	5	6	7	8	9	NA
5.2.1	Performing a menu selection operation leads to predictable results	Never		Always	1	2	3	4	5	6	7	8	9	NA
5.3	Error messages	Unhelpful		Helpful	1	2	3	4	5	6	7	8	9	NA
5.4	Avatar controls	Unnatural		Natural	1	2	3	4	5	6	7	8	9	NA
5.4.1	Response time to most avatar related operations	Too slow		Fast enough	1	2	3	4	5	6	7	8	9	NA
5.4.2	Effort to perform most avatar related operations	Too much		Unnoticeable	1	2	3	4	5	6	7	8	9	NA
5.5	The system is reliable	Never		Always	1	2	3	4	5	6	7	8	9	NA
5.6	Comments about System / Feedback:													

**Part 6: Screen Layout**

6.1	Text characters on screen	Hard to read	Neutral	Easy to read	1	2	3	4	5	6	7	8	9	NA
6.1.1	Clarity of text characters	Fuzzy		Sharp	1	2	3	4	5	6	7	8	9	NA
6.1.2	Font legibility	Barely legible		Very legible	1	2	3	4	5	6	7	8	9	NA
6.2	Screen layouts seemed intuitive	Never		Always	1	2	3	4	5	6	7	8	9	NA
6.2.1	Amount of information that is displayed on screen	Inadequate		Adequate	1	2	3	4	5	6	7	8	9	NA
6.2.2		Illogical		Logical										

Arrangement of information displayed on screen	1	2	3	4	5	6	7	8	9	NA
6.3 Comments about Screen Layout:										

**Part 7: Visual Quality**

7.1	Quality of emoticon graphics	Bad				Neutral				Good		NA	
		1	2	3	4	5	6	7	8	9			
7.2	Implementation of 3d objects	Confusing							Clear				NA
		1	2	3	4	5	6	7	8	9			
7.2.1	Colours used are	Unnatural							Natural				NA
		1	2	3	4	5	6	7	8	9			
7.2.2	Range of colours available	Inadequate							Adequate				NA
		1	2	3	4	5	6	7	8	9			
7.3 Comments about Visual Quality													

**General Comments:**



## Appendix 2 – User Instructions

---

### *Accessing gridWorld:*

GridWorld is currently held on a private server within Edith Cowan University, School of Communication and Multimedia Sciences. Users can access the gridWorld project homepage through <http://malevich.dyndns.org/~hanwei>.

GridWorld is a Shockwave 3D program and has the following minimum requirements:

128 Mb SDRAM

800 MHz Pentium II / Power PC G4

32 Mb 3D accelerated graphics card

### *How to use gridWorld:*

- 1) To move, just click on the floor's tiles.
  
- 2) To access user menu, click on your avatar. You can perform actions on other users by clicking on their avatars.
  
- 3) Avatar's must be within your "hearing-range" before your can "hear" their messages.
  
- 4) To rotate the camera's view, hold shift and move mouse up or down.
  
- 5) Public, private and host avatars have different appearances. You must click on a host avatar before you can join a private chat.

## Appendix 3 – Lingo code

---

The gridWorld interface is run by 48 lingo scripts, separated into four casts each handling key chat operations, with an additional 23 chatML scripts to support MU functionality.

Of the four script cast, the “Script” cast is the largest, containing 11 behaviour scripts, 7 parent scripts, and 10 movie scripts supporting the majority of gridWorld’s 3D environment and object controls, point-and-click interactivity and other interface and chat operations.

The lingo code for the “Script” cast script members are outlined in the following pages:

- 1) Behaviour Scripts
- 2) Parent Scripts
- 3) Movie Scripts

## A2.1. "Script" Cast: Behaviour Scripts

### Behaviour Script: 1 – loop

```
on exitFrame me
  go to the frame
end
```

### Behaviour Script: 2 – exitFrame.Follow

```
global s3d -- the member of the 3d sprite
global thisSprite -- the spriteNum of the 3d sprite
global userAvatar -- user's avatar object
-----
-- Frame behaviour script attached to 3d sprite to make
user
-- avatar follow a selected avatar
-- created: 24/7/2002
-----
on exitFrame

  sendSprite (thisSprite, #getMouseOver)-- update position
of the pointer
  --
  if userAvatar.avFollow = True then
    -- get user position
    isectData = s3d.modelsUnderRay(userAvatar.userPos,
vector(0,-1,0), 1, #detailed)
    -- find avatar name to follow
    n = " avatar.bottom " & userAvatar.followName

    if voidP(s3d.model(n)) then -- check if target avatar
still exists
      userAvatar.avFollow = False
      userAvatar.followName = "none"
    else -- target avatar exists
      -- send user avatar position and target avatar
position to user avatar's movement scripts
      userAvatar.moveToObj.setAvTrajectory(isectData,
s3d.model(n).worldposition)
    end if
  end if
  --
  if userAvatar.followName <> "none" then -- double check
if user is still following someone
    userAvatar.avFollow = True
  end if

  go to the frame - 5 -- repeat every five frames
```

```
end exitFrame
```

### Behaviour Script: 3 – exitFrame.getMouseOver

```
global thisSprite -- the spriteNum of the 3d sprite
global userAvatar -- user's avatar object
-----
-- Frame behaviour script used to update position of the
pointer
-- created: 24/7/2002
-----
on exitFrame
  sendSprite (thisSprite, #getMouseOver)-- update position
of the pointer
end exitFrame
```

### Behaviour Script: 4 – toggleCamera

```
global defaultCameraVector -- the initial camera vector
global tagH -- height at which nametag will hover over
the avatar

property bMode -- button mode
-----
-- Button behaviour used to toggle camera view from
-- third person to birds eye view. The script uses
-- a binary toggle switch to change its functionality.
--
-- The behaviour is initially set to mode 2 (i.e. the
-- camera is in its initial, third person vector) setting
-- the camera to toggle up to birds eye view when
clicked.
-----
on beginSprite(me)
  bMode = 2 -- initial "ground" view, behaviour set to
mode 2
end beginSprite

-----
on mouseUp(me)
  case bMode of
    1: -- behaviour set to mode 1
      bMode = 2 -- set behaviour to mode 2
      toggleCameraDown() -- toggle "ground" view
    2: -- behaviour set to mode 1
      bMode = 1 -- set behaviour to mode 1
      toggleCameraUp() -- toggle "birds-eye" view
  end case
```

```
end mouseUp
```

```
-----  
on toggleCameraUp(me) -- set the camera to jump to bird's  
eye view  
  member("gridWorld").camera[1].transform.position =  
vector(-0,1000,0) -- bird's eye vector  
  xPoint = member("gridWorld").model(" dummy  
").worldPosition.x  
  zPoint = member("gridWorld").model(" dummy  
").worldPosition.z  
  member("gridWorld").camera[1].pointAt(xPoint,0,zPoint)  
-- move camera  
  tagH = 26 -- correct nametag hover height  
end toggleCameraUp
```

```
-----  
on toggleCameraDown(me) -- set the camera to jump to  
original view  
  member("gridWorld").camera[1].transform.position =  
defaultCameraVector -- initial vector  
  xPoint = member("gridWorld").model(" dummy  
").worldPosition.x  
  zPoint = member("gridWorld").model(" dummy  
").worldPosition.z  
  member("gridWorld").camera[1].pointAt(xPoint,0,zPoint)  
-- move camera  
  tagH = 100 -- correct nametag hover height  
end toggleCameraDown
```

### **Behaviour Script: 5 - button.Login**

```
-----  
-- Button behaviour script to jump to "chat" marker  
-----  
on mouseUp(me)  
  go to "chat"  
end
```

## Behaviour Script: 6 - button.sendMessage

```
-----  
-- Button behaviour script to call sendMessage handler  
-----  
on mouseUp me  
    sendMessage()  
end
```

## Behaviour Script: 7 - behaviour.updateTimer

```
global avatarList  
  
property secTimer -- seconds counter  
property minTimer -- minute counter  
property m -- "00" minute value  
-----  
---  
-- Behaviour script attached to 3d sprite to update the  
chat  
-- clock which will be used for measuring avatar  
statistics.  
-----  
---  
on beginSprite(me)  
    -- pre/reset clock values  
    secTimer = 0  
    minTimer = 0  
    m = "00"  
end beginSprite  
  
-----  
---  
on exitFrame(me)  
    updateAvatarTimer() -- update clock for avatar-  
statistics  
end exitFrame  
  
-----  
---  
on updateAvatarTimer(me)  
    secTimer = secTimer + 0.03 -- increment seconds  
  
    if secTimer > 30 and secTimer < 30.03 then  
        repeat with i = 1 to avatarList.count -- run through  
all active avatars in gridWorld  
            avatarList[i].statObj.updateBlend() -- update  
short-term effects  
        end repeat
```

```

else if secTimer >60 then
  repeat with i = 1 to avatarList.count -- run through
all active avatars in gridWorld
    avatarList[i].statObj.updateScale() -- update an
avatar's flag size
    avatarList[i].statObj.updateBlend() -- update
short-term effects
  end repeat
  secTimer = 0 -- reset seconds
  minTimer = minTimer + 1 -- increment minutes

  if minTimer < 10 then
    m = "0"&minTimer
  else
    m = minTimer
  end if
end if

  if secTimer < 10 then -- write the minute value into
clock
    member("timer").text = "[ "& m &" : 0"& secTimer & "
]"
  else
    member("timer").text = "[ "& m &" : "& secTimer & "
]"
  end if
end updateAvatarTimer

```

## Behaviour Script: 8 - behaviour.emoteCheckBox

```
global emote
-----
-- Behaviour script attached to emoticon
-- selection menus.
-- created:20/9/2002
-----
on beginSprite(me)
    emote = "none" -- preset emoticon menu selection to
    "none" expression
end beginSprite

-----
on mouseUp(me)
    -- return radio button selected
    selected = sendAllSprites (#RadioGroup_SelectedButton,
    "emoteGroup")

    case selected.name of -- set emoticon to emote
        "none": emote = "none"
        "happy": emote = "happy"
        "wink": emote = "wink"
        "supprise": emote = "supprise"
        "interest": emote = "interest"
        "fear": emote = "fear"
        "sad": emote = "sad"
        "angry": emote = "anger"
        "disgust": emote = "disgust"
        "neutral": emote = "neutral"
    end case

end mouse Up

-----
on toggleSelf(me, checked)
    if checked <> sprite(me.spriteNum).member.name then
        sprite(me.spriteNum).member.hilite = False
    else
        sprite(me.spriteNum).member.hilite = True
        emote = checked
    end if
end toggleSelf
```



## Behaviour Script: 9 - behaviour.chatBox

```
global gNet
global userAvatar
-----
-- Behavior script attached to chat input box.
-----
on keyUp(me)
  if the keycode = 36 then -- check to see if enter has
  been pressed
    delete member("chatField").line[2] -- erase carriage
  return
  sendMessage() -- call to movie handler to send
  message
  else
    -- Thinking animation cycle is off or has ended
    if userAvatar.animateObj.thinkCycle = 0 or
    userAvatar.animateObj.thinkCycle > 5 then
      id = userAvatar.userName
      c = member("chatField").text.chars.count
      if c >= 0 and c < 20 then -- short message length
        zScale = 10
      else if c >= 20 and c < 40 then -- medium message
      length
        zScale = 20
      else if c >= 40 and c < 60 then -- long message length
        zScale = 30
      else if c >= 60 then -- maximum message length
        zScale = 40
      end if

      set s = [:]
      s.addprop("command", "Custom Event")
      s.addprop("target", "@AllUsers")
      s.addprop("event", "think")
      s.addprop("id", id)
      s.addprop("scale", zScale)
      gNet.send(s)
    else
      -- Thinking animation still go, but message length
      has hit milestone
      c = member("chatField").text.chars.count
      if c = 20 then -- message length has grown past
      short length
        zScale = 10
        set s = [:]
        s.addprop("command", "Custom Event")
        s.addprop("target", "@AllUsers")
        s.addprop("event", "think")
        s.addprop("id", id)
        s.addprop("scale", zScale)
        gNet.send(s)
```

```

        else if c = 40 then -- message length has grown
past medium length
            zScale = 20
            set s = [:]
            s.addprop("command", "Custom Event")
            s.addprop("target", "@AllUsers")
            s.addprop("event", "think")
            s.addprop("id", id)
            s.addprop("scale", zScale)
            gNet.send(s)
        else if c = 40 then -- message length has grown
past long length
            zScale = 30
            set s = [:]
            s.addprop("command", "Custom Event")
            s.addprop("target", "@AllUsers")
            s.addprop("event", "think")
            s.addprop("id", id)
            s.addprop("scale", zScale)
            gNet.send(s)
        else if c = 60 then -- message length has hit
maximum length
            zScale = 40
            set s = [:]
            s.addprop("command", "Custom Event")
            s.addprop("target", "@AllUsers")
            s.addprop("event", "think")
            s.addprop("id", id)
            s.addprop("scale", zScale)
            gNet.send(s)
        end if
    end if
end if
end keyup

```

## Behaviour Script: 10 - behaviour.history

```
global s3d
global gNet -- global chatML object

property phistory -- history member
property hHeight -- height of history image
property hWidth -- width of history image
-----
-- Behaviour script attached to history sprite.
-----
on beginSprite(me)
    pHistory = member("history")
    pHistory.image.fill(0,0,700,180, rgb(192,220,192))
    hHeight = phistory.height
    hWidth = phistory.width
end beginSprite

-----
on drawPage(me, tImage, nameTag, tTopLeft, emote)
    tRect = tImage.rect -- calculate the rect of the
message image
    pImage = member("history").image -- get copy of old
history image

    targetY = hHeight - tRect.height - 5
    targetX = tTopLeft.LocH --(tRect.width/2)
    tHeight = targetY + tRect.height
    tWidth = targetX + tRect.width

    -- update history
    pHistory.image.copyPixels(pImage, rect(0,-tRect.height-
20,700,180-tRect.height-20), pImage.rect)
    pImage.fill(0,targetY-16,700,180, rgb(192,221,192))
    -- draw nametag and message
    pHistory.image.copyPixels(nameTag, rect(targetX-
5,targetY-16,tWidth-5,targetY), nameTag.rect)
    if emote <> "none" then -- redraw for no emoticon
        pHistory.image.copyPixels(member(emote).image,
rect(targetX,targetY-1,targetX +
member(emote).image.rect.right,targetY-1 +
member(emote).image.rect.height),
member(emote).image.rect,
[#maskImage:member(emote).image.createMask(), #ink:36])
        pHistory.image.copyPixels(tImage, rect(targetX +
member(emote).image.rect.right + 2,targetY,tWidth +
member(emote).image.rect.right,tHeight), tRect)
    else -- redraw for emoticon included
        pHistory.image.copyPixels(tImage,
rect(targetX,targetY,tWidth,tHeight), tRect)
    end if
```

```
end drawPage
```

### Behaviour Script: 11 - behaviour.gridWorld

```
global thisSprite
global s3d
global gNet
global worldReady
global worldObj -- 3d world object
global userAvatar -- user avatar object
global avatarList -- list of user's avatars
global tileSize -- size of grid
global avColour -- avatar's Colour
global avShape -- avatar's Shape
global tagH -- height of name tage above avatar

property isectData -- list returned by modelsUnderLoc()
function
property setPoint -- target vector for pointer position
property onSprite -- property to check if mouse is within
the 3d sprite
property tempString
-----
-- Behaviour script attached to the 3d sprite. Controls
-- behaviour handlers for gridWorld
-----
on beginSprite(me)
  createAlpha() -- used to create new text box alphas

  thisSprite = sprite(1) -- set 3d sprite object
  s3d = thisSprite.member -- set 3d member object
  onSprite = False
  tagH = 100
  avatarList = []
  isectData = []

  userAvatar = new(script
"avatar.parent"(member("field.userName").text),avColour,a
vShape)
  worldObj = new(script "gridWorld.parent")

  worldObj.createWorld() -- calls createGrid() and
initializeValues()
  userAvatar.createAvatar(userAvatar)

  userAvatar.setAvatar()
  userAvatar.chatObj.setChat(userAvatar)

  worldObj.createDummy() -- calls and sets dummy model
member("followAv").text = userAvatar.followName
```

```

worldReady = True

sound(2).queue([#member: member("chatTag")]) -- cue
sound effect in ram
end beginSprite

-----
on mouseUp(me)
  if the doubleClick then -- check for double click
    doubleClickedAction()
  else
    startTimer
    repeat while the timer < 10
      if the mouseDown then -- user has double clicked
        doubleClickedAction()
        exit
      end if
    end repeat
    getMouseClicked()
  end if
end mouseUp

-----
on mouseWithin(me)
  set onSprite = True -- mouse within 3D s'prite
end mouseWithin

-----
on mouseLeave(me)
  set onSprite = False -- mouse left 3D s'prite
end mouseLeave

-----
-----
-- getMouseOver handler controls the mouse
rollover/pointer
-- tracking operations withing the 3D chat environment.
--
-- The getMouseOver handler will not fully execute if
onSprite
-- property is False.
-- created: 23/7/2002
on getMouseOver()

  if onSprite = True then -- mouse within 3d s'prite
    -- find where the mouse clicked...
    pointWithinSprite = the mouseLoc -
point(thisSprite.left, thisSprite.top)
    isectData =
s3d.camera(1).modelsUnderLoc(pointwithinSprite,
#detailed)

    if isectData <> [] then -- pointing at something

```

```

-- get clicked on model's name
modelName = string(isectData[1].model)
-- check if user clicked on the floor model
case modelName.word[2] of
  "gridWorld":
    t = getProp(isectData[1],#vertices) -- return
vector location of user avatar
    tempString = t[2]
    vect1 = tempString.x--word[2] -- extract x
vector value
    vect3 = tempString.z--word[4] -- extract z
vector value
    setPoint =
vector(float(vect1),userAvatar.userRadius,float(vect3)) +
vector(-(tileSize/2),0,tileSize/2)
  end case
  if the shiftDown then -- rotate camera mode
    worldObj.pointer.visibility = #none
    worldObj.pointer.transform.position = setPoint -
vector(0,userAvatar.userRadius + 0.5,0)
  else -- normal pointer mode
    worldObj.pointer.visibility = #both
    worldObj.pointer.transform.position = setPoint -
vector(0,userAvatar.userRadius - 0.5,0)
  end if
  else -- pointing at nothing
    worldObj.pointer.visibility = #none
  end if
  else -- mouse outside of 3d sprite
    worldObj.pointer.visibility = #none
  end if

```

```
end getMouseOver
```

```
-----
---
```

```
-- getMouseClicked tracks the mouse clicks user makes
within
```

```
-- the 3D chat environment.
```

```
-- modified: 15/7/2002
```

```
on getMouseClicked(me)
```

```
-- check if clicked on any models
```

```
if isectData.count <> 0 then -- clicked on something
```

```
-- get clicked-on model's name
```

```
modelName = string(isectData[1].model)
```

```
-- determine action to be taken
```

```
case modelName.word[2] of
```

```
  "pointer": -- clicked on pointer
```

```
    -- call to move avatar
```

```
    userAvatar.moveToObj.setAvTrajectory(isectData,
setPoint)
```

```
    jumpMarkerMovement()
```

```
        "gridWorld": -- clicked on floor
        otherwise: -- clicked on avatar
            whichAvatar(modelName)
        end case
    else -- clicked on nothing
        return
    end if
end getMouseClicked
```

```
-----
on doubleClickedAction()
    -- return to normal movement mode
    userAvatar.followName = "none"
    userAvatar.avFollow = False
    member("followAv").text = "none"
end doubleClickedAction
```

```
-----
on exitFrame
    -- update ALL avatars in gridWorld
    repeat with i = 1 to avatarList.count
        avatarList[i].checkAvatar1()
    end repeat
end exitFrame
```

## A2.2. "Script" Cast: Parent Scripts

### Parent Script: 1 – gridWorld.parent

```
global thisSprite -- sprite(me.spriteNum)
global s3d -- thisSprite.member
global avToSpritePos -- avatar's position relative to 3d
sprite position
global avatarList
global tileSize -- size of tile
global defaultCameraVector

property worldCam -- world camera 1
property gridWorld -- "Floor" object
property dummy -- dummy object for camera focus
property pointer -- plane to denote where pointer is on
"Floor"
property pointShader -- pointer shader
property gridShader -- floor shader
property gridMap -- image file
property worldWidth -- width of the floor
property worldLength -- length of the floor
-----
-- Parent script for creation of gridWorld's floor,
camera,
-- lighting, textures, and user avatar.
-----
on new(me)
  return(me) -- return object
end new

-----
-- Private Functions --
-----
on createWorld(me)
  -- reset 3d world
  s3d.resetWorld() -- reset 3d world
  --
  defaultCameraVector = vector(-250,550,-500)

  -- call functions --
  initializeValues()
  createShaders()
  createGrid()
  calculateTiles()
  setTexture()
  createPointer()

  -- create textures for thinking animation
  noDots =
  s3d.newTexture("0dot",#fromCastMember,member("0-dots"))
```



```

noDots.nearFiltering = false
noDots.quality = #medium
oneDots =
s3d.newTexture("1dot",#fromCastMember,member("1-dots"))
oneDots.nearFiltering = false
oneDots.quality = #medium
twoDots =
s3d.newTexture("2dot",#fromCastMember,member("2-dots"))
twoDots.nearFiltering = false
twoDots.quality = #medium
threeDots =
s3d.newTexture("3dot",#fromCastMember,member("3-dots"))
threeDots.nearFiltering = false
threeDots.quality = #medium

-- set up directional lighting
s3d.newLight("userDirectional",#directional)
s3d.light[3].transform.rotation = vector(-145,0,0)
--
end createWorld

```

```

-----
-- initializeValue(): last modified: 15/7/2002
on initializeValues(me)

```

```

the floatPrecision = 1
tileSize = 50
worldWidth = 700
worldLength = 1000

```

```

end initializeValues

```

```

-----
-- set shader values
on createShaders(me)
pointShader = s3d.newShader("pointerShader",#standard)
gridShader = s3d.newShader("worldShader",#standard)
--
pointShader.ambient = rgb(50,50,50) -- set ambient
pointShader.diffuse = rgb(50,100,200) -- set diffuse
pointShader.shininess = 0 -- set shininess off
pointShader.texture = void -- assign a void texture to
pointShader
pointShader.flat = true -- use flat shading
pointShader.blend = 20
--
gridShader.shininess = 0
gridShader.renderStyle = #fill
gridShader.texture = void
gridShader.blend = 100

end createShaders

```

```

-----
on createPointer(me)
  -- create plane resource
  pResource = s3d.newModelResource(" plane ",#plane)
  pResource.length = tileSize
  pResource.width = tileSize
  -- create pointer model
  pointer = s3d.newModel(" pointer ",pResource)
  pointer.transform.rotate(-90,0,0)
  -- add shader
  pointer.shaderList[1] = pointShader
  pointer.shaderList[2] = pointShader
  pointer.visibility = #none
end createPointerPlane

```

```

-----
-- createCrid(): last modified: 15/7/2002
on createGrid()
  -- Create new plane resource
  gResource = s3d.newModelResource("gridMap", #plane)
  gResource.length = worldLength -- plane model's yAxis
size
  gResource.width = worldWidth -- plane model's xAxis
size
  gResource.lengthVertices = (worldLength/tileSize + 1)
  gResource.widthVertices = (worldWidth/tileSize + 1)

  -- create 'gridWorld' model from plane resource
  gridWorld = s3d.newModel(" gridWorld ",gResource)
  -- orientate the "ground"
  gridWorld.rotate(90,0,0) -- rotate the floor to lie
horizontal
  gridWorld.transform.translate(worldWidth /
2,0,worldLength / 2) -- set top-left of tile to
vector(0,0,0)

  -- set shaders
  gridWorld.shaderList[1] = gridShader
  -- gridWorld.shaderList[2] = gridShader
  -- set visibility
  gridWorld.visibility = #front
  -- set inker
  gridWorld.addModifier(#inker)

end createGrid

```

```

-----
on setTexture(me)
  --assign texture to gridShader
  gridMapTexture = s3d.newTexture("gridTexture")
  gridMapTexture.image = gridMap

```

```

    gridShader.diffuseLightMap =
s3d.newTexture("base",#fromCastMember,member("grid-
floor_small"))
    gridShader.texture = gridMapTexture -- assign texture
map to gridShader

    gridShader.blend = 50
    gridShader.textureModeList[2] = #none
    gridShader.blendFunctionList[1] = #add
    gridShader.blendFunctionList[2] = #blend
    gridShader.blendConstantList[2] = 20
    -- tweeking... --
    gridMapTexture.quality = #low
    gridMapTexture.nearFiltering = False

end setTexture

-----
-- calculateTiles(): last modified: 15/7/2002
on calculateTiles(me)
    -- Calculate number of square tiles needed
    tileNumWidth = worldWidth / tileSize -- number of tiles
across
    tileNumLength = worldLength / tileSize -- number of
tiles down

    rect1 = 0
    rect2 = 0
    rect3 = tileSize
    rect4 = tileSize

    gridMap = image(worldWidth, worldLength, 32) --
initialize gridmap image
    repeat with i = 0 to tileNumLength -- row
        rect1 = 0
        rect3 = tileSize
        repeat with p = 0 to tileNumWidth -- column
            gridMap.copyPixels(member("newTile").image,
rect(rect1,rect2,rect3,rect4), member("newTile").rect)
            rect1 = rect1 + tileSize
            rect3 = rect3 + tileSize
        end repeat
        rect2 = rect2 + tileSize
        rect4 = rect4 + tileSize
    end repeat
    alphaMap = image(worldWidth, worldLength, 8)
    alphaMap.copyPixels(gridMap,gridMap.rect,gridMap.rect)

-- member("tmpGrid").image = gridMap
gridMap.setAlpha(255)
gridMap.setAlpha(alphaMap)
gridMap.useAlpha = true

```

```
end calculateGrid
```

```
-----  
on createDummy()  
  -- create dummy object  
  dummy = s3d.newModel(" dummy ")  
  dummy.Resource = s3d.newModelResource("dummy", #plane)  
  dummy.Resource.width = 1  
  dummy.Resource.length = 1  
  dummy.transform.position.y = -1  
  dummy.visibility = #none  
  setCamera(dummy)  
end createDummy
```

```
-----  
on setCamera(dummyObj)  
  -- camera positioning  
  worldCam = thisSprite.camera(1)  
  worldCam.transform.position = defaultCameraVector --  
vector(-100,400,350)  
  worldCam.pointAt(0,0,0)  
  -- camera properties  
  worldCam.hither = 20  
  worldCam.projectionAngle = 40  
  -- set camera as child of avatar  
  dummy = dummyObj  
  dummy.addChild(worldCam)  
end setCamera
```

```
-----  
on moveCamera(me, num)  
  -- follow avatar with camera  
  dummy.transform.position =  
avatarList[num].userModelPoint.worldPosition  
end moveCamera
```

## Parent Script: 2 – avatar.parent

```
global thisSprite
global s3d
global avatarList -- avatar model list
global worldObj -- gridWorld.parent object
global tileSize -- size of grid squares
global tagH -- height of name tage above avatar

property chatObj -- chat.parent object
property moveToObj -- moveTo.parent object
property collisionObj -- collision.parent object
property statObj -- avStats.parent object
property animateObj -- animation parent object

property userModel -- avatar "feet"
property userModelTop -- avatar "body"
property userModelPoint -- avatar parent/collision
bounding box
property hBounds -- hearing boundary

property userName -- user name
property userRadius -- radius of avatar sphere
property userPos -- user's world vector coordnents
property userNumber -- user number based on
avatarList.count
property userTalk -- whether an avatar has a chat box or
not

property avShader -- shader of avatar
property avShader1
property avShader2
property avatarColour -- colour of avatar
property avatarShape -- shape of avatar

property talkCounter -- number of frames past since
message display
property talkCount -- flag to count frames: true/false

property overlayIndexBack -- index of first overlay
property overlayIndexFore -- index of second overlay
property overlayRect -- rect obj to overlay text image
property backTexture -- texture that is used to display
text backdrop
property foreTexture -- texture that is used to display
text messages
property historyText

property avToSpritePos -- avatar's locH and locV
```

```

property avatarGo -- status of avatar's movement
property avSpeed -- speed of avatar
property myAnglePerUnit -- angle to turn per unit
traveled

property avThink -- true/false value of when an avatar is
in "thinking" mode
property avFollow -- true/false value of when an avatar
is in "following" mode
property followName -- name of user to follow

property avChatStatus -- status of user in chat world
property channelHost -- name of channel's host
property channelName -- name of channel (if exist)
-----
-----
-----
on new(me, name, avColour, avShape)

    -- void
    me.userModel = void
    me.userPos = void
    me.backTexture = void
    me.foreTexture = void
    me.chatObj = void
    me.moveToObj = void
    me.followName = "none"
    me.historyText = []
    me.avChatStatus = "public"
    me.channelHost = "none"
    me.channelName = "none"

    me.overlayIndexBack = -1
    me.overlayIndexFore = -1

    -- Values
    me.userName = name
    me.avatarColour = avColour
    me.avatarShape = avShape

    -- True/False
    me.talkCount = False
    me.userTalk = False
    me.avatarGo = False
    me.avThink = False
    me.avFollow = False

    -- Vectors
    me.overlayRect = rect(0,0,0,0)

    -- Integers
    me.userNumber = -1

```

```

me.talkCounter = 0
me.userRadius = 15 -- set radius
me.avSpeed = 5
me.myAnglePerUnit = 180 / (userRadius * pi) -- rotate
angle
return me
end

```

```

-----
-----
-- createAvatar(): modified 19/7/2002 --
on createAvatar(userAvatar)

-- set avatarNumber
avatarList.add(userAvatar) -- add avatar Object to
global avatar list
userNumber = avatarList.count -- assign a user number
to avatar object (note: user always number 1)

-- generate model resource
aList = createResource(avatarShape, userRadius,
userName)

-- create 'avatar' model from sphere and box resource
userModel = s3d.newModel(" avatar.bottom "& userName &
",aList[1])
userModelTop = s3d.newModel(" avatar.top "& userName &
",aList[2])
userModelPoint = s3d.newModel(" avatar.point "&
userName & " ",aList[3])
hBounds = s3d.newModel(" hearRange "& userName & "
",aList[4])

createModel(aList, userRadius, userName, userModel,
userModelTop, userModelPoint, hbounds)

-- create shaders for models
avShader = s3d.newShader("avShader
"&userName,#standard)
avShader1 = s3d.newShader("avShader.t
"&userName,#standard)
avShader2 = s3d.newShader("avShader.h
"&userName,#standard)

setAvShader(avatarColour, userName, avShader,
avShader1, avShader2, userModel, userModelTop,
userModelPoint, hbounds)

-- set control objects
chatObj = new(script "chat.parent"(userNumber))
moveToObj = new(script "moveTo.parent"(userNumber))

```

```

collisionObj = new(script
"collision.parent"(userNumber))
statObj = new(script "avStats.parent"(userNumber))
animateObj = new(script "animation.parent"(userNumber))

-- debug control
userModel.debug = false
userModelTop.debug = false
userModelPoint.debug = false

-- attach collision detection modifiers
collisionObj.setCollisionBoundary(userModelPoint,
"#bumper")
collisionObj.setCollisionBoundary(hBounds, "#listen")

-- initialise overlays
chatObj.iniOverlay()

--initial test of avatar
checkAvatar1()

end createAvatar

```

```

-----
-----

```

```

-- Custom Handlers -----

```

```

-----
-----

```

```

on setAvatar()
  userPos = userModel.worldPosition
  avatarGo = False
  isectData = s3d.modelsUnderRay(userPos, vector(0,-1,0),
1, #detailed)
  setPoint = vector((tileSize/2),userRadius,(tileSize/2))
  moveToObj.setAvTrajectory(isectData, setPoint)
end setAvatar

```

```

-----
-----

```

```

on avPosToSpritePos()
  avToSpritePos =
sprite(1).camera.worldSpaceToSpriteSpace(userPos)
end avPosToSpritePos

```

```

-----
-----

```

```

on startClock()
  -- Reset talk timer
  talkCount = True
  userTalk = True

```



```
    talkCounter = 0
end startClock
```

```
-----
-----
on stopClock()
  -- Stop talk timer
  talkCount = False
  userTalk = False
  talkCounter = 0
  chatObj.chatLocArea = [170,170,170]
end stopClock
```

```
-----
-----
on checkAvatar1()
  userPos = userModel.worldPosition -- update Avatar's
world coordinants
  --
  if avatarGo then
    moveToObj.avMoveToPosition()
  end if
  --
  checkAvatar2()

end checkAvatar1
```

```
-----
-----
on checkAvatar2()
  if talkCount = True then
    animateObj.speachPopUp(talkCounter)
    talkCounter = talkCounter + 1 -- count
    -- check if message has been onscreen for 5 seconds
    if (talkCounter > 30 * 5) then
      -- chatObj.hideOverlay(overlayIndexBack) --
hide overlay1
      chatObj.hideOverlay(overlayIndexFore) -- hide
overlay2
      stopClock() -- stop and reset timer
    end if
  end if
  --
  --
  if avThink = True then
    animateObj.thinkingAvatar()
  end if
  --
  checkAvatar3()

end checkAvatar2
```

```
-----
-----
on checkAvatar3()
```

```

-- update avatar's screen position
avPosToSpritePos()

-- align the overlays to the avatar
if avToSpritePos <> VOID then
  if userTalk = True then
    chatObj.updateOverlay(overlayIndexFore) -- update
all overlays
  end if
  thisSprite.camera.overlay[overlayIndexBack].loc =
point(avToSpritePos.locH - 64, avToSpritePos.locV - tagH)
  thisSprite.camera.overlay[overlayIndexBack].scale = 1
-- nameTag visible on overlay 1
  else
    -- make overlay's invisible
    thisSprite.camera.overlay[overlayIndexBack].scale = 0
-- nameTag invisible on overlay 1
    thisSprite.camera.overlay[overlayIndexFore].scale = 0
-- chat message invisible on overlay 2
  end if
  --
  userModelPoint.collision.enabled = False -- disable
collision detection
  collisionObj.myFrame = 0 -- set to 0 so collision will
execute next frame
end checkAvatar3
-----
-----
--on stepFrame me
--  checkAvatar1()
--  checkAvatar2()
--  checkAvatar3()
--end stepFrame

```

### Parent Script: 3 - avStats.parent

```
global s3d
global thisSprite
global avatarList -- global list of avatars

property avatarObj -- avatar object
property avScale
property postRecord
property timeRecord
property hPost
property pCount -- average Post-rate
-----
-- Parent script for tracking and updating avatar
-- statistics.
-----
on new(me, userNum)
  avatarObj = avatarList[userNum]
  --
  me.avScale = avatarObj.userModelTop
  me.postRecord = [20,1]
  me.timeRecord = [0,0]
  me.hPost = [20,1]
  --
  me.pCount = 0
  --
  return me
end new
-----
---
--on longTermStatus(me)
--  postRecord = []
--end longTermStatus
-----
---
--on shortTermStatus(me)
--  updateScale()
--  updateBlend()
--end shortTermStatus
-----
---
on updateScale(me)
  avMsgLength = postRecord[1]/postRecord[2] -- average
  message length

  if avMsgLength <= 5 then -- low chatting
    p = 0.60
  else if avMsgLength > 5 and avMsgLength <= 10 then --
  low-medium chatting
    p = 0.70
```

```

    else if avMsgLength > 10 and avMsgLength <= 15 then --
medium chatting
        p = 0.80
    else if avMsgLength > 15 and avMsgLength <=20 then --
medim-high chatting
        p = 0.90
    else if avMsgLength > 20 and avMsgLength <=30 then --
high chatting
        p = 1.00
    else if avMsgLength > 30 then -- very high chatting
        p = 1.10
    end if

    if hPost = postRecord then
        postRecord = [postRecord[1]+0, postRecord[2]+1]
    end if

    hPost = postRecord
    -- change avScale
    avScale.transform.scale =
vector(p,p,avScale.transform.scale.z)

end updateScale
-----
---
on updateBlend(me)
    if pCount <= 1 then
        if avatarObj.avShader.blend > 10 +
            put avatarObj.userName
            s3d.shader("avShader.t "& avatarObj.userName).blend
= s3d.shader("avShader.t "& avatarObj.userName).blend -
10
            --          avatarObj.avShader1.blend =
avatarObj.avShader1.blend - 10
        end if
    else if pCount > 1 then
        if avatarObj.avShader1.blend < 90 then
            s3d.shader("avShader.t "& avatarObj.userName).blend
= s3d.shader("avShader.t "& avatarObj.userName).blend +
10
            --          avatarObj.avShader1.blend =
avatarObj.avShader1.blend + 10
        end if
    end if
    put pCount
    put s3d.shader("avShader.t "& avatarObj.userName).blend
    pCount = 0
end updateBlend

```

## Parent Script: 4 – moveTo.parent

```
global thisSprite
global s3d
global worldObj -- proprietary machine-user object
global avatarList -- list of avatars
global tileSize -- size of grid squares
global gNet -- MU object

property avatarObj -- user object
property pointerPos -- pointer position vector (relative
to world coordinates)
property targetPos -- center of tile clicked on
property avTrajectory -- avatar's movement vector
property travelDist -- distance for avatar to travel
property iniPoint -- start vector of avatar's travels
property tempString -- temporary string value for
targetPos

-----
-- Parent script for controlling and tracking avatar
-- movements on the chat floor.
-----

on new(me, userNum)
  avatarObj = avatarList[userNum]
  return me
end new

-----
---
on setAvTrajectory(me, isectData, setPoint)
  if isectData <> [] then
    pointerPos = isectData[1].isectPosition -- position
mouse clicked
    targetPos = setPoint
    -- calculate trajectory vector for avatar
    avTrajectory = (targetPos - avatarObj.userPos) -- get
travel vector
    iniPoint = avatarObj.userPos
    travelDist = avTrajectory.magnitude -- get travel
vector's magitude

    avTrajectory.y = 0 -- keep avatar on ground
    avTrajectory = avTrajectory / travelDist -- caculate
avatar's trajectory vector

    -- prepare movement paramaters
    moveList = []
```

```

    moveList =
[avatarObj.userName,iniPoint,travelDist,targetPos,avTrajectory]
    -- prepare to send
    s = [:]
    s.addprop("command", "Custom Event")
    s.addprop("target", "@AllUsers")
    s.addprop("event", "move")
    s.addprop("mList", moveList)
    -- send info
    gNet.send(s)
end if

end setAvTrajectory

-----
---
-- modified: 19/7/2002
on avMoveToPosition(me)
    currentVect = (avatarObj.userPos - iniPoint)
    currentDist = currentVect.magnitude

    if currentDist >= travelDist then
        avatarObj.avatarGo = False -- stop avatar moving
        avatarObj.userModelPoint.transform.position =
targetPos

        -- animate avatar
        animateCount = 0
        avatarObj.animateObj.resetAvatar()
    else
        -- Move Avatar

avatarObj.userModelPoint.transform.translate(avTrajectory
* avatarObj.avSpeed)

        -- Make the ball roll as it moves find the axis of
rotation
        axis = avTrajectory.crossProduct(vector(0, -1, 0))
        angleTurned = avTrajectory.magnitude *
avatarObj.myAnglePerUnit * avatarObj.avSpeed
        ballPosition = avatarObj.userModel.worldPosition

        -- Rotate the ball around its center relative to the
world
        avatarObj.userModel.rotate(ballPosition, axis,
angleTurned, #world)

        if avatarObj.userName = member("field.userName").text
then
            worldObj.moveCamera(avatarObj.userNumber) -- move
dummy to move camera
        end if

```

```
    -- animate avatar
    avatarObj.animateObj.animateAvatar(avTrajectory) --
animate the avatar
    end if

end avMoveToPosition
```

## Parent Scripts: 5 – chat.parent

```
global thisSprite
global s3d
global avatarList -- list of avatars
global userAvatar -- proprietary machine-user object

property avatarObj -- proprietary machine-user object
property chatRect -- rect(0,0,0,0)
--property backRect -- background overlay rect
--property foreRect -- foreground overlay rect
property scaleValue -- scale value for Overlay
property blendValue -- blend value for Overlay
property chatTexture
property chatLocArea -- area occupied by message overlay
-----
-- Parent script for controlling chat message operations,
-- chat message rendering, etc.
-----
on new(me, userNum)
    me.avatarObj = avatarList[userNum]
    me.chatRect = []
    me.scaleValue = 0
    me.blendValue = 30
    me.chatLocArea = [170,170,170] -- top, bottom,
centerline

    return me
end

-----
on setChat(me)
    -- CHAT.INI --
    member("chatText").text = " "
    member("chatField").text = "Hello world!"
    member("chatField").editable = True
    -- avatarObj.foreTexture.image =
member("chatText").image

thisSprite.camera.overlay[avatarObj.overlayIndexFore].scale = 0

end setChat

-----
on iniOverlay(me)
    -- initiate overlay textures
    chatTexture = s3d.newTexture("chatBox "&
avatarObj.userName,#fromCastMember,member("llnBox"))
```



```

-- create overlay textures
avatarObj.backTexture = s3d.newTexture("NameTag " &
avatarObj.userName) -- texture for backdrop overlays
avatarObj.foreTexture = chatTexture -- texture for
frontdrop overlays

-- attach overlays to camera
thisSprite.camera.addOverlay(avatarObj.backTexture,
point(0,0), 0) -- set backTexture as first private
overlay
thisSprite.camera.addOverlay(avatarObj.foreTexture,
point(0,0), 0) -- set foreTexture as second private
overlay

-- set overlayCount
avatarObj.overlayIndexFore =
thisSprite.camera.overlay.count
avatarObj.overlayIndexBack = avatarObj.overlayIndexFore
- 1

createNameOverlay

end iniOverlay

```

```

-----
-- modified: 26/7/2002
on createOverlay(me, message, emote)
member("chatText").text = message -- write message
string into temporary cast member
avatarObj.historyText[1] = message
avatarObj.historyText[2] = emote
member("chatText").paragraph[1].color =
avatarObj.avatarColour
-- update statObj values
avatarObj.statObj.postRecord[1] =
avatarObj.statObj.postRecord[1] +
member("chatText").text.length
avatarObj.statObj.postRecord[2] =
avatarObj.statObj.postRecord[2] + 1
avatarObj.statObj.pCount = avatarObj.statObj.pCount + 1

chatImage = member("chatText").image --
image(128, (avatarObj.overlayRect.bottom) +
chatRect.bottom, 8)
member("tmpchat").image = chatImage--.extractAlpha()
repeat with i = 4 to 216
if member("tmpchat").image.getPixel(i, 0) = rgb( 255,
255, 255 ) then
chatWidth = i-1
put chatWidth
exit repeat
end if
end repeat

```

```

chatRect[1] = chatImage.rect.left --+ 4
chatRect[2] = chatImage.rect.top-- + 4
chatRect[3] = chatImage.rect.right --+ 4
chatRect[4] = chatImage.rect.bottom --+ 4

i = member("chatText").height
if i <=16 then -- 1 line message
  if chatWidth <= 150 then
    chatRect[3] = chatWidth
    member("1lnBox-half").image = member("1line-
half").image
    b = "1lnBox-half"
  else
    member("1lnBox").image = member("1line").image --
redraw background
    b = "1lnBox"
  end if
  n = 3
  else if i>16 and i<48 then -- 2 line message
    member("2lnBox").image = member("2line").image --
redraw background
    b = "2lnBox"
    n = 3
  else if i>=48 then -- 3 line message
    member("3lnBox").image = member("3line").image --
redraw background
    b = "3lnBox"
    n = 3
  end if
  -- draw emoticon
  if emote <> "none" then
    drawFace(emote, b)
    n = 24
  end if
  -- draw message
  member(b).image.copyPixels(chatImage, rect(chatRect[1]
+ n,chatRect[2],chatRect[3] + n,chatRect[4]),
rect(chatRect[1],chatRect[2],chatRect[3],chatRect[4]),
[#maskImage:chatImage.createMask(), #ink:36])
  chatTexture.image = member(b).image -- update message
box image

s3d.camera[1].overlay[avatarObj.overlayIndexFore].source
= chatTexture -- update overlay texture
  avatarObj.startClock() -- start timing message lifespan
end

-----
on drawFace(emote, b)
  faceImage = member(emote).image
  fRect = []
  fRect[1] = faceImage.rect.left

```

```

fRect[2] = faceImage.rect.top
fRect[3] = faceImage.rect.right
fRect[4] = faceImage.rect.bottom

member(b).image.copyPixels(faceImage,
rect(fRect[1]+2,fRect[2]+2,fRect[3]+2,fRect[4]+2),
rect(fRect[1],fRect[2],fRect[3],fRect[4]),
[#maskImage:faceImage.createMask(), #ink:36])

end drawFace

-----

on createNameOverlay(me)
  member("nameTag").text = avatarObj.userName
  member("nameTag").alignment = #center
  member("nameTag").forecolor =
value(avatarObj.avatarColour.paletteIndex)
  avatarObj.backTexture.image = member("nameTag").image
end createNameOverlay

-----

on hideOverlay(me, overlay)
  thisSprite.camera.overlay[overlay].scale = 0 -- set the
overlay size to 0
  drawHistory()
end hideOverlay

-----

on drawHistory(me)
  if (avatarObj.avToSpritePos <> VOID) then
    -- check if message is within hearing range
    if (avatarObj.userName = userAvatar.userName) or
(blendValue = 100) then
      tTopLeft = calcChatOverlay()
      member("nameTag").text = avatarObj.userName
      member("nameTag").alignment = #left
      member("nameTag").forecolor =
value(avatarObj.avatarColour.paletteIndex)

      member("historyText").text =
avatarObj.historyText[1]
      member("historyText").forecolor =
value(avatarObj.avatarColour.paletteIndex)

      sendSprite(3, #drawPage,
member("historyText").image, member("nameTag").image,
tTopLeft, avatarObj.historyText[2])
      end if
    end if

end drawHistory

-----

```

```

on updateOverlay(me,overlay2)
  -- calculate the position of the avatar's overlay
  tTopLeft = calcChatOverlay()
  h = chatRect[4]-chatRect[2]
  t = tTopLeft.locV
  b = tTopLeft.locV + h
  c = tTopLeft.locV + (h/2)
  chatLocArea[1] = t
  chatLocArea[2] = b
  chatLocArea[3] = c

  if avatarObj.userName <> userAvatar.userName then
    repeat with i = 1 to (avatarObj.userNumber - 1)
      if avatarList[i].userTalk = True then
        if c > avatarList[i].chatObj.chatLocArea[3] then
          if t <= avatarList[i].chatObj.chatLocArea[2]
then
            tTopLeft.locV =
avatarList[i].chatObj.chatLocArea[2] + 5
            --
            avatarList[i].chatObj.chatLocArea[2] =
tTopLeft.locV
            --
            end if
          else
            if b >= avatarList[i].chatObj.chatLocArea[1]
then
              tTopLeft.locV =
avatarList[i].chatObj.chatLocArea[1] - (h+4)
              --
              avatarList[i].chatObj.chatLocArea[1] =
tTopLeft.locV
              --
              end if
            end if
          end if
        end repeat

thisSprite.camera.overlay[avatarObj.overlayIndexFore].blend = blendValue -- set blend of foregroundOverlay
end if

-- align the overlays to the avatar
thisSprite.camera.overlay[overlay2].loc = tTopLeft
-- reset blend values
blendValue = 30
end

-----
on calcChatOverlay(me)
  tTopLeft = point(avatarObj.avToSpritePos.locH - 64, \

```

```
avatarObj.avToSpritePos.locV -  
(avatarObj.overlayRect.bottom) - 25)  
    return tTopLeft  
end calcChatOverlay
```

## Parent Script 6 – collision.parent

```
global thisSprite
global s3d
global gNet
global tileSize
global avatarList -- list of avatars
global userAvatar -- proprietary machine-user object

property avatarObj -- userObject
property myFrame -- the current frame
-----
-- Parent scripts for collision detections.
-----
on new(me, userNumber)
    avatarObj = avatarList[userNumber]
    return me
end new

-----
on setCollisionBoundary(me, cModel, handle)
    cModel.addmodifier(#collision)
    cModel.collision.mode = #mesh
    cModel.collision.resolve = False
    cModel.collision.setCollisionCallback(value(handle),
me)
end setCollisionBoundary

-----
on bumper(me, collisionData)
    -- 0) check that collision will only detect once per
frame by
    -- ignoring the second collision.
    -- * may become obsolete in future versions of Director
    if myFrame = the Frame then
        exit
    end if
    myFrame = the Frame

    -- 1) check if model who did the colliding is the one
who called the handler
    if collisionData.modelA <> avatarObj.userModelPoint
then
        exit
    else
        -- 2) check if the model who called the handler is
moving or stationary.
        if avatarObj.avatarGo = false then
            exit
        else
```

```

        if avatarObj.moveToObj.targetPos =
collisionData.modelB.worldPosition then

            isectData = s3d.modelsUnderRay(avatarObj.userPos,
vector(0,-1,0), 1, #detailed)
            --          modelName = string(isectData[1].model)

            tempString = isectData[1].vertices[2]
--          tempString = tempString.item[4..6]

            vect1 = tempString.x--.word[2] -- extract x
vector value
--          vect1 = vect1.item[1] -- get rid of comma
            vect3 = tempString.z--.word[4] -- extract z
vector value
            setPoint =
vector(float(vect1),avatarObj.userRadius,float(vect3)) +
vector(-(tileSize/2),0,tileSize/2)

            avatarObj.moveToObj.setAvTrajectory(isectData,
setPoint)

            end if
        end if
    end if
end bumper

```

```

-----
on listen(me, collisionData)

```

```

    avatarObj.userModelPoint.collision.enabled = True
    -- check if model who did the colliding is the one who
called the handler
    if collisionData.modelA = userAvatar.hBounds then --
user called handler
        if string(collisionData.modelB).word[3] =
userAvatar.followName.word[1] then
            userAvatar.avFollow = False
        end if
    else -- other avatar called handler
        if string(collisionData.modelB).word[3] =
userAvatar.userName then
            avatarObj.chatObj.scaleValue = 1
            avatarObj.chatObj.blendValue = 100
        end if
    end if
    -- end if
end listen

```

## Parent Script: 7 ~ animation.parent

```
global s3d
global thisSprite
global avatarList -- global list of avatars
--
property avatarObj -- avatar object
property animateCount -- counter for timing "walking"
animation
property thinkCount -- counter for timing "thinking"
animation
property thinkCycle -- counter for "thinking" cycles
--
property zScale
property yRotation
-----
-- Parent script for controlling avatar animation cycles.
-----
on new(me, userNum)
  avatarObj = avatarList[userNum]
  me.animateCount = 0 -- counter to 0
  me.thinkCount = 0 -- counter to 0
  me.thinkCycle = 0 -- counter to 0
  me.zScale = 10
  me.yRotation = 0
  return me
end new

-----
on speachPopUp(me, i)
  case i of
    -- open message box
    0: n = 0
    1: n = 0.2
    2: n = 0.4
    3: n = 0.6
    4: n = 0.8
    5: n = 1
    if avatarObj.userNumber = 1 or
    avatarObj.chatObj.blendValue = 100 then
      sound(2).play(member("chatTag"))
    end if

    -- close message box
    146: n = 0.8
    147: n = 0.6
    148: n = 0.4
    149: n = 0.2
    150: n = 0
    -- else
    otherwise: n = 1
```



```

end case

-- make overlay's visible
-- thisSprite.camera.overlay[overlayIndexBack].scale =
1 -- nameTag visible on overlay 1

thisSprite.camera.overlay[avatarObj.overlayIndexFore].sca
le = n -- chat message invisible on overlay 2
-- Reset transparency

thisSprite.camera.overlay[avatarObj.overlayIndexFore].ble
nd = 100 -- set blend of foregroundOverlay

end speachPopUp

-----
-- created: 19/7/2002
on animateAvatar(me, avTrajectory)
  case animateCount of
    0:
      avatarObj.userModelTop.transform.rotation =
vector(0,0,0)
      avatarObj.userModelTop.transform.position =
vector(0,30,0)
    1:
avatarObj.userModelTop.rotate(avTrajectory*2,#parent)
      -- 3:
avatarObj.userModelTop.rotate(avTrajectory*1.5,#parent)
    3: avatarObj.userModelTop.rotate(-
avTrajectory*2,#parent)
    5: avatarObj.userModelTop.rotate(-
avTrajectory*2,#parent)
      -- 5: avatarObj.userModelTop.rotate(-
avTrajectory*1.5,#parent)
    7:
avatarObj.userModelTop.rotate(avTrajectory*2,#parent)
      -- 7:
avatarObj.userModelTop.rotate(avTrajectory*1.5,#parent)
    9:
avatarObj.userModelTop.rotate(avTrajectory*1.5,#parent)

  end case
  -- increment counter
  animateCount = animateCount + 1
  -- check counter
  if animateCount > 7 then
    set animateCount = 1
  end if

end animateAvatar

-----
on thinkingAvatar(me)

```

```

yRotation = yRotation + 5
if yRotation > 359 then
    yRotation = 0
end if

. if thinkCycle > 5 then
--     if avatarObj.userModelTop.transform.scale.x < 1.0
then
--         avatarObj.userModelTop.transform.scale =
vector(1.0,1.0,1.0)
--     else
        avatarObj.userModelTop.transform.scale =
vector(avatarObj.userModelTop.transform.scale.x,avatarObj
.userModelTop.transform.scale.y,1.0)
--     end if
        avatarObj.userModelTop.transform.rotation =
vector(0,0,0)
        avatarObj.avShader1.texture = s3d.texture("name
"&avatarObj.userName)
        avatarObj.avThink = False
        thinkCount = 0
        thinkCycle = 0
    else
        case thinkCount of
            0:
                avatarObj.userModelTop.transform.scale =
vector(avatarObj.userModelTop.transform.scale.x,avatarObj
.userModelTop.transform.scale.y,zScale)
                avatarObj.avShader1.texture = s3d.texture("0dot")
            10:
                avatarObj.userModelTop.transform.scale =
vector(avatarObj.userModelTop.transform.scale.x,avatarObj
.userModelTop.transform.scale.y,zScale)
                avatarObj.avShader1.texture = s3d.texture("1dot")
            20:
                avatarObj.userModelTop.transform.scale =
vector(avatarObj.userModelTop.transform.scale.x,avatarObj
.userModelTop.transform.scale.y,zScale)
                avatarObj.avShader1.texture = s3d.texture("2dot")
            30:
                avatarObj.userModelTop.transform.scale =
vector(avatarObj.userModelTop.transform.scale.x,avatarObj
.userModelTop.transform.scale.y,zScale)
                avatarObj.avShader1.texture = s3d.texture("3dot")
                thinkCycle = thinkCycle + 1
        end case
        -- increment counter
        thinkCount = thinkCount + 1
        -- check counter
        if thinkCount > 40 then
            set thinkCount = 0
        end if
    end if
end if

```

```
    avatarObj.userModelTop.transform.rotation =  
vector(0,yRotation,0)  
    end if
```

```
end thinkingAvatar
```

```
-----  
on resetAvatar(me)  
    avatarObj.userModelTop.transform.rotation =  
vector(0,0,0)  
    avatarObj.userModelTop.transform.position =  
vector(0,30,0)  
end resetAvatar
```

## A2.3. "Script" Cast: Movie Scripts

### Movie Script: 1 – movie.create-remove

```
global thisSprite
global s3d
global userAvatar
global message
global avatarList -- avatar model list

-----
-- Public Methods --
-----
-- Movie script containing handlers for creating
-- and removing avatars.
-----
on createAnotherAvatar(plist)
  plist = value(plist)
  -- create obj of "avatar.parent" called Jim
  av = new(script "avatar.parent"(plist[1]))
  av.avatarColour = plist[2]
  av.avatarShape = plist[9]

  -- create new user-sphere resource
  av.createAvatar()
  -- av.setAvatar()
  av.moveToObj.iniPoint = plist[3]
  av.moveToObj.travelDist = plist[4]
  av.moveToObj.targetPos = plist[5]
  av.moveToObj.avTrajectory = plist[6]
  av.avatarGo = plist[7]
  av.userPos = plist[8]
  av.avChatStatus = plist[12]
  av.channelHost = plist[13]
  av.channelName = plist[14]

  checkChannel(av)

  s3d.shader("avShader.t "& plist[1]).blend = plist[10]
  s3d.shader("avShader.h "& plist[1]).blend = plist[11]

  av.userModelPoint.transform.position = av.userPos

  av.chatObj.createOverlay("hi mom!", "happy")

  return av
end

-----
-----
on checkChannel(av)
```

```

if av.avChatStatus = "Host" then
  prepareHostStatus(av)
else if av.avChatStatus = "private" then
  preparePrivateStatus(av)
else if av.avChatStatus = "public" then
  preparePublicStatus(av)
end if
end checkChannel

-----
on removeAvatar(avatarName)
  -- remove user avatar model, modelResource, shaders,
  overlays, textures, etc
  repeat with p = 1 to avatarList.count
    if avatarList[p].userName = avatarName then
      i = p
    end if
  end repeat

  if avatarList[i].userName = avatarName then
    -- remove from actorList
    -- (the actorList).deleteOne(avatarList[i])

    -- delete the avatar models
    s3d.deleteModel(" avatar.point "& avatarName & " ")
    s3d.deleteModel(" avatar.top "& avatarName & " ")
    s3d.deleteModel(" avatar.bottom "& avatarName & " ")
    s3d.deleteModel(" hearRange "& avatarName & " ")

    -- delete the avatarResources
    s3d.deleteModelResource("top."& avatarName)
    s3d.deleteModelResource("feet."& avatarName)
    s3d.deleteModelResource("mbounding."& avatarName)
    s3d.deleteModelResource("hbounding."& avatarName)
    -- remove overlay
    overlay1 = avatarList[i].overlayIndexBack
    overlay2 = avatarList[i].overlayIndexFore
    s3d.camera(1).removeOverlay(overlay2)
    s3d.camera(1).removeOverlay(overlay1)
    -- increment user overlay and number down by 1
    repeat with n = (i+1) to avatarList.count
      avatarList[n].userNumber = i -- increment
userNumber down 1
      avatarList[n].overlayIndexFore = i*2
      avatarList[n].overlayIndexBack = (i*2)-1
    end repeat
    -- removeShaders
    s3d.deleteShader("avShader "& avatarName)
    s3d.deleteShader("avShader.t "& avatarName)
    s3d.deleteShader("avShader.h "& avatarName)
    -- delete textures
    s3d.deleteTexture("name "& avatarName)
    s3d.deleteTexture("NameTag "& avatarName)
  end if
end on removeAvatar

```

```
s3d.deleteTexture("chatBox "& avatarName)
-- remove user avatar from avatar list
avatarList.deleteAt(i)
end if
end removeAvatar
```

```
-----
on getAvDetails()
  pList = []
  pList[1] = userAvatar.userName
  pList[2] = userAvatar.avatarColour
  pList[3] = userAvatar.moveToObj.iniPoint
  pList[4] = userAvatar.moveToObj.travelDist
  pList[5] = userAvatar.moveToObj.targetPos
  pList[6] = userAvatar.moveToObj.avTrajectory
  pList[7] = userAvatar.avatarGo
  pList[8] = userAvatar.userPos
  pList[9] = userAvatar.avatarShape
  pList[10] = userAvatar.avShader1.blend
  pList[11] = userAvatar.avShader2.blend
  pList[12] = userAvatar.avChatStatus
  pList[13] = userAvatar.channelHost
  pList[14] = userAvatar.channelName
  return pList
end getAvDetails
```

## Movie Script: 2 – movie.sendMessage

```
global emote
global gNet
global userAvatar
```

```
-----
-- Public Methods --
-----
```

```
-----
-- Handler for sending chat messages
-----
```

```
on sendMessage()
  -- Primary Avatar chat --
  message = member("chatField").text
  member("chatField").text = "" -- clear chat field
  pList = []
  pList = [userAvatar.userName, message]

  if userAvatar.channelName = "none" then
    targetUsers = "@AllUsers"
  else
    targetUsers = userAvatar.channelName
  end if

  set s = [:]
  s.addprop("command", "Chat")
  s.addprop("target", "@AllUsers")
  gNet.send(s)

  set s = [:]
  s.addprop("command", "Chat")
  s.addprop("target", targetUsers)
  s.addprop("message", message)
  s.addprop("emote", emote) -- emoteFace)
  gNet.send(s)

  member("chatField").editable = true
  sendAllSprites(#toggleSelf, "none")
end
```

### Movie Script: 3 – movie.jumpMarkerControl

```
global currentMarker
global userAvatar
global gNet
global avatarList

-----
-- Movie Scripts controlling marker jumping. Toggles
variables
-- to prepare values to be used at new marker
-----

on jumpMarkerMovement()
  if currentMarker = "menu" then
    -- user is currently in menu
    if member("chatStatus").text = "public" then
      go to "chat"
    else if member("chatStatus").text = "private" then
      go to "chat.private"
    else if member("chatStatus").text = "debug" then
      go to "chatDebug"
    end if
  else if currentMarker = "system_working" then
    -- system is busy
  end if

end jumpMarkerMovement()

-----

on whichAvatar(modelName)
  avName = modelName.word[3]
  repeat with i = 1 to avatarList.count
    if avatarList[i].userName = avName then
      member("name").text = ""
      member("name").text = avName --& " "
      avatarObj = avatarList[i]
      exit repeat
    end if
  end repeat

  if currentMarker <> "system_working" then
    if avName <> userAvatar.userName then
      if avatarObj.avChatStatus = "host" and
userAvatar.channelName = "none" then
        member("channelName").text =
avatarObj.channelName
        go to "otherMenu.host"
      else if userAvatar.followName =
avatarObj.userName&" " then
        go to "otherMenu.follow"
      else
```



```

        go to "otherMenu"
    end if

else
    if userAvatar.avChatStatus = "host" then
        go to "userMenu.host"
    else if userAvatar.avChatStatus = "Private" then
        go to "userMenu.private"
    else
        go to "userMenu"
    end if
end if
end if

end whichAvatar

-----
on avChangeStatus(pStatus)
    if pStatus = "host" then
        set s = [:]
        s.addprop("command", "Custom Event")
        s.addprop("target", "@AllUsers")
        s.addprop("event", "becomeHost")
        s.addprop("id", userAvatar.userName)
        s.addprop("channelName", userAvatar.channelName)
        gNet.send(s)
    else if pStatus = "private" then

    else if pStatus = "public" then
    end if
end avChangeStatus

```

## Movie Script: 4 – movie.createResource

```
-----  
-- Movie script containing handlers to build avatar  
-- flag shapes  
-----
```

```
on buildShape01(me)  
  tVertexList = [\br/>vector(12.5,0,0.5),\  
vector(12.5,75,0.5),\  
vector(-12.5,75,0.5),\  
vector(-12.5,0,0.5),\  
vector(-12.5,0,-0.5),\  
vector(-12.5,75,-0.5),\  
vector(12.5,75,-0.5),\  
vector(12.5,0,-0.5) \  
]  
  
  tFaceList = [\br/>[1,2,4],\  
[2,3,4],\  
[5,6,8],\  
[6,7,8] \  
] -- each face vertex needs to be defined in counter  
clockwise direction  
  
  mList = [tFaceList,tVertexList]  
  return mList  
  
end buildShape01
```

```
-----  
on buildShape02(me)  
  tVertexList = [\br/>vector(0,37.5,0.5), \  
vector(0,4,0.5), \  
vector(12.5,0,0.5), \  
vector(7,37.5,0.5), \  
vector(12.5,75,0.5), \  
vector(0,65,0.5),\  
vector(-12.5,75,0.5), \  
vector(-7,37.5,0.5), \  
vector(-12.5,0,0.5), \  
vector(0,37.5,-0.5),\  
vector(0,4,-0.5), \  
vector(-12.5,0,-0.5), \  
vector(-7,37.5,-0.5), \  
vector(-12.5,75,-0.5), \  
vector(0,65,-0.5), \  
vector(12.5,75,-0.5), \  
vector(7,37.5,-0.5), \  
]
```

```

vector(12.5,0,-0.5)\
]

    tFaceList = [\
[1,2,3],\
[1,3,4],\
[1,4,5],\
[1,5,6],\
[1,6,7],\
[1,7,8],\
[1,8,9],\
[1,9,2],\
[10,11,12],\
[10,12,13],\
[10,13,14],\
[10,14,15],\
[10,15,16],\
[10,16,17],\
[10,17,18],\
[10,18,11]\
] -- each face vertex needs to be defined in counter
clockwise direction

    mList = [tFaceList,tVertexList]
    return mList

end buildShape02

-----
on buildShape03(me)
    tVertexList = [\
vector(0,37.5,0.5), \
vector(0,0,0.5), \
vector(12.5,10,0.5), \
vector(12.5,37.5,0.5), \
vector(12.5,65,0.5), \
vector(0,75,0.5),\
vector(-12.5,65,0.5), \
vector(-12.5,37.5,0.5), \
vector(-12.5,10,0.5), \
vector(0,37.5,-0.5),\
vector(0,0,-0.5), \
vector(-12.5,10,-0.5), \
vector(-12.5,37.5,-0.5), \
vector(-12.5,65,-0.5), \
vector(0,75,-0.5), \
vector(12.5,65,-0.5), \
vector(12.5,37.5,-0.5), \
vector(12.5,10,-0.5)\
]

    tFaceList = [\
[1,2,3],\

```

```

[1,3,4],\
[1,4,5],\
[1,5,6],\
[1,6,7],\
[1,7,8],\
[1,8,9],\
[1,9,2],\
[10,11,12],\
[10,12,13],\
[10,13,14],\
[10,14,15],\
[10,15,16],\
[10,16,17],\
[10,17,18],\
[10,18,11]\
] -- each face vertex needs to be defined in counter
clockwise direction

```

```

mList = [tFaceList,tVertexList]
return mList

```

```

end buildShape03

```

```

-----
on buildShape04(me)
  tVertexList = [\
vector(0,37.5,0.5), \
vector(0,0,0.5), \
vector(6,20,0.5), \
vector(12.5,37.5,0.5), \
vector(6,55,0.5), \
vector(0,75,0.5), \
vector(-6,55,0.5), \
vector(-12.5,37.5,0.5), \
vector(-6,20,0.5), \
vector(0,37.5,-0.5), \
vector(0,0,-0.5), \
vector(-6,20,-0.5), \
vector(-12.5,37.5,-0.5), \
vector(-6,55,-0.5), \
vector(0,75,-0.5), \
vector(6,55,-0.5), \
vector(12.5,37.5,-0.5), \
vector(6,20,-0.5)\
]

```

```

  tFaceList = [\
[1,2,3],\
[1,3,4],\
[1,4,5],\
[1,5,6],\
[1,6,7],\
[1,7,8],\

```

```
[1,8,9],\  
[1,9,2],\  
[10,11,12],\  
[10,12,13],\  
[10,13,14],\  
[10,14,15],\  
[10,15,16],\  
[10,16,17],\  
[10,17,18],\  
[10,18,11]\  
] -- each face vertex needs to be defined in counter  
clockwise direction  
  
    mList = [tFaceList,tVertexList]  
    return mList  
  
end buildShape04
```

## Movie Script: 5 – createAlpha

```
-----  
-- Movie script for creating alpha channels for message  
boxes  
-----  
on createAlpha  
  member("1line").image.setAlpha(255)  
  member("1line").image.setAlpha(member("1alpha").image)  
  member("1line").image.useAlpha = true  
  member("1lnBox").image = member("1line").image  
  
  member("1line-half").image.setAlpha(255)  
  member("1line-half").image.setAlpha(member("1alpha-  
half").image)  
  member("1line-half").image.useAlpha = true  
  member("1lnBox-half").image = member("1line-  
half").image  
  
  member("2line").image.setAlpha(255)  
  member("2line").image.setAlpha(member("2alpha").image)  
  member("2line").image.useAlpha = true  
  member("2lnBox").image = member("2line").image  
  
  member("3line").image.setAlpha(255)  
  member("3line").image.setAlpha(member("3alpha").image)  
  member("3line").image.useAlpha = true  
  member("3lnBox").image = member("3line").image  
  
  -- code for drawing chat floor tiles  
  -- member("newTile").image.setAlpha(255)  
  --  
member("newTile").image.setAlpha(member("alphaTile").imag  
e)  
  -- member("newTile").image.useAlpha = true  
  -- member("1lnBox").image = member("1line").image  
  
end createAlpha
```

## Movie Script: 6 – movie.avatarResource

```
global s3d
global tileSize

-----
-- Movie script for creating avatar model resources.
-----
on createResource(avatarShape, userRadius, userName)
  -- create new sphere resource
  s = s3d.newModelResource("feet."& userName, #sphere)
  s.radius = userRadius
  s.resolution = 2

  case avatarShape of
    "shape01": mList = buildShape01()
    "shape02": mList = buildShape02()
    "shape03": mList = buildShape03()
    "shape04": mList = buildShape04()
  end case

  tFaceList = mList[1]
  tVertexList = mList[2]

  -- create new mesh resource
  b = s3d.newMesh("top."&userName, tFaceList.count,
tVertexList.count)
  -- define vertices for each face in new mesh resource
  b.vertexList = tVertexList
  repeat with i = 1 to tFaceList.count
    b.face[i].vertices = tFaceList[i]
  end repeat
  -- generate normals
  b.generateNormals(#flat)
  -- build
  b.build()

  -- create box resource for parent and collision
detection
  p = s3d.newModelResource("mbounding."& userName,#box)
  p.top = False
  p.bottom = False
  p.length = tileSize - 2
  p.width = tileSize - 2
  p.height = 1

  -- create box resource for hearing range
  c =
s3d.newModelResource("hbounding."&userName,#cylinder)
  c.topCap = False
  c.topRadius = 80
```

```

    c.bottomradius = 0
    c.height = 0.5
    c.numSegments = 1
    c.resolution = 4

    aList = [s, b, p, c]
    return aList
end createResource

```

### Movie Script: 7 – movie.avatarModels

```

global s3d

-----
-- Movie script for creating avatar models.
-----
on createModel(aList, userRadius, userName, userModel,
userModelTop, userModelPoint, hbounds)

    -- set parent of avatar "body" and "feet"
    userModel.parent = userModelPoint
    userModelTop.parent = userModelPoint
    hBounds.parent = userModelPoint

    -- set parent model invisible
    userModelPoint.visibility = #none
    hBounds.visibility = #back

    -- position models
    userModelTop.transform.translate(vector(0,30,0))
    userModelPoint.transform.position.y = userRadius
    hBounds.transform.position = vector(0,-(userRadius-
0.5),0)

end createModel

```



## Movie Script: 8 – movie.avatarShaders

```
global s3d
```

```
-----  
-- Movie script for creating avatar shaders.  
-----  
on setAvShader(avatarColour, userName, avShader,  
avShader1, avShader2, userModel, userModelTop,  
userModelPoint, hbounds)  
    r = avatarColour.red -- red value  
    g = avatarColour.green -- green value  
    b = avatarColour.blue -- blue value  
    --  
    v = 10 -- increment/decrement value  
  
    -- prepare avShader  
    avShader.ambient = rgb(r,g,b)  
    avShader.diffuse = rgb(r+v,g+v,b+v)  
    avShader.shininess = 0  
    avShader.texture = void -- assign a void texture to  
paintShader  
    avShader.flat = true  
    avShader.blend = 100  
  
    -- prepare avShader1  
    avShader1.ambient = rgb(r,g,b)  
    avShader1.diffuse = rgb(r,g,b)  
    avShader1.shininess = 0  
    avShader1.texture = void -- assign a void texture to  
paintShader  
    avShader1.flat = true  
    avShader1.blend = 90  
    avShader1.textureMode = #wrapPlanar  
    --  
    nameTexture = s3d.newTexture("name  
"&userName,#fromCastMember,member("nameTexture"))  
    nameTexture.nearFiltering = false  
    nameTexture.quality = #medium  
    avShader1.texture = nameTexture  
  
    -- prepare avShader2  
    avShader2.ambient = rgb(r,g,b)  
    avShader2.diffuse = rgb(r+v,g+v,b+v)  
    avShader2.shininess = 0  
    avShader2.texture = void -- assign a void texture to  
paintShader  
    avShader2.flat = true  
    avShader2.blend = 0  
  
    -- attach shaders to models
```

```

    userModel.shaderList[1] = avShader -- sphere shader
attached
    userModelTop.shaderList[1] = avShader1 -- back shader
    userModelTop.shaderList[2] = avShader1 --  shader
    userModelTop.shaderList[3] = avShader1 --  shader
    userModelTop.shaderList[4] = avShader1 --  shader
    userModelTop.shaderList[5] = avShader1 --  shader
    userModelTop.shaderList[6] = avShader1 --  shader

-- attach shader to hBounds
hBounds.shaderList[1] = avShader2

-- add #inker modifier
userModel.addmodifier(#toon)
-- userModel.inker.creases = True
userModel.inker.silhouettes = True

userModelTop.addmodifier(#inker)
-- userModelTop.inker.creases = True
userModelTop.inker.silhouettes = True
userModelTop.visibility = #front

hBounds.addmodifier(#inker)
-- hBounds.inker.creases = False
hBounds.inker.silhouettes = True
end setAvShader

```

## Movie Script: 9 – movie.chatControl

```
global s3d
global userAvatar
global gNet
-----
on prepareHostStatus(avatarObj)
  s3d.shader("avShader.h "& avatarObj.userName).blend =
50
  if avatarObj.userName <> userAvatar.userName then
    avatarObj.userModelTop.removeModifier(#inker)
    avatarObj.userModelTop.addModifier(#toon)
  end if
  avatarObj.avChatStatus = "host"
  avatarObj.channelHost = avatarObj.userName
end prepareHostStatus

-----
on preparePrivateStatus(avatarObj)
  -- s3d.shader("avShader.h "& avatarObj.userName).blend
= 50
  if avatarObj.userName <> userAvatar.userName then
    avatarObj.userModelTop.removeModifier(#inker)
  end if
  avatarObj.hBounds.shaderList[1] =
s3d.shader("avShader.h "& avatarObj.channelHost)
end preparePrivateStatus

-----
on preparePublicStatus(avatarObj)
  -- check if toon modifier exists
  if avatarObj.userModelTop.modifier.count = 1 then
    avatarObj.userModelTop.removeModifier(#toon)
  end if

  -- change inker modifier back to normal mode
  avatarObj.userModelTop.addModifier(#inker)
  avatarObj.hBounds.shaderList[1] = avatarObj.avShader2
  avatarObj.avShader2.blend = 0

  -- reset avatar chat values
  avatarObj.avChatStatus = "public"
  avatarObj.channelHost = "none"
  avatarObj.channelName = "none"

  -- if avatarObj.channelName <> "none" then

  -- end if

end preparePrivateStatus
```

## Movie Script: 10 – movie.chatProcesses

```
global s3d
global gNet
global avatarList
global userAvatar
global worldReady

global custom

-----
-- Movie script controlling the sending and receiving
-- of chat and avatar information using gNet (chatML
-- object).
-----
on processNetStuff(me, m)

    set c = m.find("command").value
    if worldReady = True then
        case c of

            "Chat": -- chat procedure
                repeat with i = 1 to avatarList.count
                    if m.find("from").value =
avatarList[i].userName then
                        -- avatarList[i].stopClock()
                        -- create the message

                        if avatarList[i].userTalk = True then

avatarList[i].chatObj.hideOverlay(avatarList[i].overlayIn
dexFore)
                            end if

avatarList[i].chatObj.createOverlay(m.find("message").val
ue, m.find("emote").value)
                                avatarList[i].animateObj.thinkCycle = 10

                            end if
                        end repeat

            "Custom Event":
                set custom = m.find("event").value
                case custom of
                    "messageSent": -- user has finished responding
                        repeat with i = 1 to avatarList.count
                            if m.find("from").value =
avatarList[i].userName then
                                    avatarList[i].animateObj.thinkCycle = 10
                                end if
                            end repeat
                end case
        end case
    end if
end on
```

```

"adduser": -- adduser procedure
  put "adding user..."
  pList = value(m.find("list").value)
  put "addUser: "&pList
  member("UserList").text = pList[1] & return &
member("UserList").text

  repeat with i = 1 to avatarList.count
    -- check if avatar already exists
    if (pList[1] = avatarList[i].userName) then
      exit -- dont add user
    end if
  end repeat

  -- add user
  av = createAnotherAvatar(pList)
  av.setAvatar()

  -- generate new pList to send to
respondAddUser
  pList = getAvDetails()
  set s = [:]
  s.addprop("command", "Custom Event")
  s.addprop("target", "@AllUsers")
  s.addprop("event", "respondadduser")
  s.addprop("list", pList)
  put "sending request to respondAddUser: "&s
  gNet.send(s)

command
"respondadduser": -- respond to an adduser

  put "responding..."
  avList = value(m.find("list").value)
  -- avExist = False
  put "respondAddUser: "& avList
  repeat with i = 1 to avatarList.count
    if (avList[1] = avatarList[i].userName)
then
      exit -- avExist = True
    end if
  end repeat

  -- if (avExist = False) then
  member("UserList").text = avList[1] & return
& member("UserList").text
  createAnotherAvatar(avList)
  --end if

"removeuser": -- kill user
  uName = m.find("name").value
  removeUser(uName)

"move": -- move avatar's user

```

```

mList = value(m.find("mList").value)
moveMe(mList)

"think": -- user is typing a message
  id = string(m.find("id").value)
  zScale = value(m.find("scale").value)
  repeat with i = 1 to avatarList.count
    if avatarList[i].userName = id then
      avatarList[i].avThink = True
      avatarList[i].animateObj.thinkCycle = 0
      avatarList[i].animateObj.zScale = zScale
      exit repeat
    end if
  end repeat

"killingChannel":
  s = [:]
  s.addprop("command", "Leave Channel")
  s.addprop("channel", userAvatar.channelName)
  gNet.send(s)

  s = [:]
  s.addprop("command", "Custom Event")
  s.addprop("target", "@AllUsers")
  s.addprop("event", "becomePublic")
  s.addprop("targetID", userAvatar.userName)
  gNet.send(s)

"becomeHost": -- change to host status
  id = string(m.find("id").value)
  repeat with i = 1 to avatarList.count
    if avatarList[i].userName = id then
      avatarList[i].channelName =
string(m.find("channelName").value)
      prepareHostStatus(avatarList[i])
      exit repeat
    end if
  end repeat

"becomePrivate": -- change to private status
  id = string(m.find("id").value)
  repeat with i = 1 to avatarList.count
    if avatarList[i].userName = id then
      avatarList[i].avChatStatus = "private"
      avatarList[i].channelHost =
string(m.find("hostName").value)
      avatarList[i].channelName =
string(m.find("channelName").value)
      preparePrivateStatus(avatarList[i])
      exit repeat
    end if
  end repeat
end repeat

```

```

    "becomePublic": -- change back to public status
        id = string(m.find("targetID").value)
        if id = "all" then
            preparePublicStatus(userAvatar)
        else
            repeat with i = 1 to avatarList.count
                if avatarList[i].userName = id then
                    preparePublicStatus(avatarList[i])
                    exit repeat
                end if
            end repeat
        end if

    end case
end case
end if

end

-----
-- Custom Handlers --
-----

on moveMe(mList)
    repeat with i = 1 to avatarList.count
        if mList[1] = avatarList[i].userName then
            avatarList[i].moveToObj.iniPoint = mList[2]
            avatarList[i].moveToObj.travelDist = mList[3]
            avatarList[i].moveToObj.targetPos = mList[4]
            avatarList[i].moveToObj.avTrajectory = mList[5]
            avatarList[i].avatarGo = True
        end if
    end repeat

end moveMe

-----

on removeUser(uName)
    set t = member("UserList").text
    set n = uName
    set i = 1
    set c = t.line.count

    repeat while i = 1 to c
        if (t.line[i] = n) then
            delete t.line[i]
            member("UserList").text = t
            removeAvatar(n)
            return
        end if
    end repeat
end removeUser

-----

```

