Design Notations for Creating Virtual Environments

Charlene Elliott, Gary Marsden, Marion Walton & Edwin Blake
Department of Computer Science
University of Cape Town
gaz@cs.uct.ac.za

ABSTRACT
In this paper we propose a new design notation to improve communication in teams creating virtual environments (VEs). Our experience in creating VEs is that the programmers and designers have no common formalisms which results in ambiguity and misunderstanding in creating the final VE. After teaching a selection of specification techniques to design students, we realized that we needed to create our own formalism. This we used with designers who found the notation useful and intuitive. More importantly, the programmers were able to interpret the formalism more accurately and reduce the time required to create virtual environments.

Author Keywords
Virtual Reality, Visual Formalism, Design Notation.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Within the Computer Science Department at the University of Cape Town, we were engaged in a effort to produce a low-cost virtual reality platform. The broad goal was to make the medium more accessible by lowering the costs involved (both hardware and software) and designing a methodology and associated tools to streamline the creation of virtual environments. It is the design of that methodology with which this paper is concerned.

To test the system from end to end, we selected three application scenarios which required a virtual reality solution; to be deemed satisfactory, the methodology and software was required to support all the requirements demanded by those scenarios.

The first scenario was related to culture preservation and a virtual environment was created in which school children could explore oral histories. A second scenario was created to educate HIV positive mothers about the impact and importance of diet [**Error! Reference source not found.**]. Allocated to both these scenarios was a dedicated team of systems programmers who created the digital resources required by the researchers. This involved not just the creation of the virtual environment, but supporting and extending the underlying authoring tool to make sure that it was as complete as possible. By the end of the second scenario, we had an authoring tool that provided a rich set of facilities for creating almost any virtual environment.

The third scenario, however, was not designed to test the functionality of the tool, but, rather, the *process* by which a user could create their own virtual environment (VE). The goal was not to create an end-user tool such as an earlier version of Alice [1], whereby a domain expert could create their own virtual environment. This was because we observed that the domain experts we worked with from the previous applications (e.g. HIV/AIDS councilors) were unlikely to have the time required to learn an authoring tool. Instead, we wanted to design a system that would allow these domain experts to create a specification that would then allow a programmer to create the desired environment without having to improvise details or becoming a content expert in their own right. The challenge of the third scenario can be stated as *creating a specification process that was suitably intuitive for a domain expert to use whilst being sufficiently unambiguous for a programmer to implement*. At the outset of the process it was not clear to us if the result of the third scenario would be a 'How To' book; a piece of software that guided the process, or some combination of the two.

It is the meeting of this challenge of communication between domain experts and VE implementers that is the subject of this paper.

SPECIFYING VIRTUAL ENVIRONMENTS
VE design and creation is a relatively new field and as such there is not yet much depth to the theoretical and empirical research in the literature. Fewer still have studied supporting the communication between designer and programmer. The design and creation of VEs is a complex process resulting in a small number of proposed methodologies to simplify the process ([4,5,6,7,8]). Schwartz et al [5] describe their experiences in creating a shared distributed VE application called "The Virtual Playground – Netgate Mall" in which designers made use of many specification techniques. They used written narratives, concept sketches, storyboards, pattern languages

and UML case-based diagrams to document a design for a Virtual Playground. The introduction of pattern languages, taken from architectural spatial planning design was a novel contribution to help plan the spatial organization of the VE and specifically to promote social space between the multiple users of the VE. While it might be true that team members were happy to use the techniques they chose, the authors did not discuss how this influenced the understanding of the design between team members. We believe that VE creation in which the designers and programmers could choose artifacts already familiar to one of the groups would lean towards unnecessary communication discrepancies. This is because each technique has its own set of rules and terminology that is not shared. This will require both designers and programmers to learn the rules and terms of any particular artifact chosen, to the same level, before effective communication can take place.

Fencott [6], Tanriverdi and Jacob [8], and Kaur [4] outline a more structured approach to VE creation. In their research they attempt to suggest step-by-step phases in which the design evolves from one state to the next before it is finally implemented. By defining a methodology, they hope to make VE creation easily grasped and understood by designers and programmers.

Based on Kaur's [4] VE creation methodology, Fencott [6] proposes a VE creation methodology with an emphasis on modeling the *intended user experiences*. Designers can benefit from his model in that it attempts to structure the design and provide a mechanism to account for the user's experiences.

He suggests different methods for specifying the plans of the VE before the implementation takes place. Again, designers and programmers are encouraged to use many specification artifacts including perceptual maps, use-case diagrams and scene graphs, of which he invented perceptual maps. Perceptual maps are diagrams used to show the relationship between the objects and interactions in the VE and how they support the intended user experience. Designers could easily construct these maps as they simply provide categories to describe VE content in English sentences. His suggestions of use-case and scene graphs, however, would still be foreign to designers with no programming experience. While Tanriverdi and Jacob [8] focus almost exclusively on interface components Fencott [7] also suggests how the designer would account for the narrative components in the VE by means of the potential for narrative embodied in the various possible paths through the environment.

Tanriverdi and Jacob [8] suggest a design model and methodology for designers of VR interfaces. Their goal is to guide the designer in the conceptual model of the design by breaking up the task into components that can be worked on separately (such as graphics, objects and interactions). Designers are constrained to start off with a textual description of each of the components which is called the high-level phase and then iteratively produce a formal specification of the high-level descriptions. The interaction documentation consists of the use of data-flow diagrams and state chart diagrams. The formal specification of these interactions is supported with a tool called PWIMP [9], which is designed to specify non-WIMP (Windows, Icon, Menu, Pointer) interfaces, such as VE interfaces. While their methodology attempts to break the design task into different categories, their use of data-flow diagrams and state-chart diagrams is very much software engineering specific, requiring the designer to learn computer science terms and logic.

Kaur et al [4] created a hypertext tool to assist designers in specifying usability requirements for their design. The tool essentially documents usability guidelines, examples and a check list for designers to follow which is then presented in hypertext format. The designers used the tool to guide the development of storyboards for given VE scenarios. Through the use of the tool, the designers were able to document interaction support into their storyboard designs. The guidelines were based on a theoretical understanding of human-computer interaction in VEs, interaction behaviour and design requirements. Empirical testing of the tool was conducted and the results showed that the tool helped designers to uncover and improve the usability of the design and identify issues which designers may not have considered otherwise. It would be interesting to see how programmers might respond to the storyboards produced and whether the designs could easily be understood by programmers. While Kaur's tool support for VE creation positively aided the designers, the focus of the tool was to promote usability awareness and not necessarily on documenting the VE application requirements to be used in the creation phase.

The last approach we look at in VE design and creation involves the testing of both a designer and programmer creating a VE application. Cho et al [10] portray their case study involving the design of a scientific learning VE between a science teacher and programmer using an authoring tool called CLOVES. CLOVES (Construction of Layer Oriented Virtual Environments for Science Inquiry Learning) is a virtual world builder that supports the development of information-rich environments using rule-based scripting. The purpose of the case study was to establish whether the programmer and teacher could come to a shared representation of the design given that they were experts in different domains and to improve upon CLOVES. In this study the designer and programmer designed the application requirements together, making the programmer a co-designer.

The case study involved two design phases: a synopsis phase and a high-level design phase. The synopsis design required the subjects to learn how CLOVES works, to investigate the models which were available to them and to brainstorm their VE. This means that the teacher learnt

computer graphics programming terminology before designing the application and thus had a steep learning curve. After learning CLOVES the subjects mostly worked with paper, pencil and a whiteboard medium to define and document their design. The high-level design phase consisted of the programmer writing the rules for the world and then along with the designer, placing the objects and the rules into CLOVES.

This study was observed by a researcher, who was also the developer of the CLOVES authoring system. By observing the teacher-programmer team during design and implementation, he was able to identify extensions and improvements to the rule-based scripting language. These extensions were not intended to allow the designer to use the authoring tool herself, but rather to extend the tool to enable the application requirements to be implemented in the tool. The output of their study showed that the teacher could learn and understand the concepts, terms and vocabulary in order for her to understand and use the CLOVES system in a limited way. By the end of the session she had learnt new words such as "pixel", "object" and "properties" and therefore they believe it showed that a common ground could be established with the teacher and programmer. Even though the designer and programmer came up with a working scenario together, the designer was not given the tools to create the design herself. Independent design work is not supported. This case study also places a heavy burden on the programmer to become a content expert in a potentially new domain in order to fulfill the role of co-designer.

The case studies into VE design and creation shows that designers are attempting to engage in VE design. Even so, it shows that designers are limited in how they engage in VE design and creation. The current methodologies use formalisms that have been borrowed from the design processes of other media and still force the design to eventually be documented in a software engineering-specific technique. This would require both designer and programmer to learn new terminology. These short-comings need to be addressed to truly allow designers and programmers to come to a shared understanding of the design.

MAKING FRIENDS WITH FUNNY PEOPLE
The first step in our research, therefore, was to find if there were any design formalisms or artifacts that were used consistently between designers and programmers. To do this, we wanted to create project groups comprising of designers and programmers. We therefore partnered with an Interactive Media course in the Department of Film and Media. These students were familiar with more traditional forms of media (print and film) but had no experience of communication using virtual environments. These were exactly the type of people our tool was aimed at; someone who was experienced in communication, but not using this new form of media.

Methodology
The first phase of our work was to discover the ways in which the students would choose to specify the virtual environment. The students were therefore taught standard techniques, such as design documents and storyboards. We also added techniques discussed by Fencott [6,7], Tanriverdi [8], Kaur [4] and Schwartz [5], such as flow charting, use-case diagrams and pseudo-code, to see if the students would appropriate these as more fitting ways to describe interactive media. Students were also given tutorials on using Alice [1] as a way of exposing them to scripting and virtual environment authoring tools.

(It must be remembered that we are not building a tool for those who are already experts in creating design documents etc. We wanted to find the techniques that worked, so that we could embed these in software or in the 'How to' guide that would ultimately lead to the unambiguous specification of the environment.)

The students were split into groups and asked to submit documents describing how their environment should be implemented. The students were free to choose whichever techniques they felt best described the environment they wished to create, but a template design document was given to them to illustrate the types of information they should be providing. Each group was allocated a member of our research team who had the role of programmer and who took part in all meetings and discussions in order to observe how the students were rationalizing the design process. This allowed us to make ethnographic observations of the group but the programmer also acted as a consultant, helping the group understand what is possible within a virtual environment.

Observations on the Design Document
Having received the specifications from the students, as programmers, the first thing we needed to begin implementation were the models and objects required for the environment. Details of models tended to be spread throughout the length of the document: description of a texture in one place; spawning point in another place etc. Often this distribution would result in key attributes of models not being defined (spawn points were rarely mentioned). Our observers commented that in the discussions between designers floor-plans were used to overcome these problems, yet none of the groups included floor-plans in their final document submission. So when positions were described they tended to be relative; e.g. "far corner of the club". A further problem in inferring the objects is that most groups made no distinction between objects in the back-story (which did not need to be rendered) and objects to populate the environment.

Interactions between objects were also poorly specified. Again, much of the terminology was ambiguous: e.g. one character needed "enough" money to purchase equipment. Whilst key interactions were specified, often the design would leave out what the characters would be doing outside of the documented interaction or what would happen should

the interaction not follow an anticipated path: e.g. what happens when the character does not have enough money? In other words they fell into the trap of thinking linearly instead of in terms of the many branching possibilities available to the user in the VE. The interactions that were specified did indeed make some use of pseudo-code (if-then types of interaction being popular) and flowcharts, but again, these representations did not indicate what would happen should the conditions for interaction not be met. Some interactions were specified in terms of camera movements, which were unfamiliar to the programmers and, in some instances, were not possible within the confines of the environment.

Finally, to implement the designs, the shortcomings in the design document led the programmers to create their own tables to aggregate information from the document. Creating these tables was a slow process, that required much interaction with the designers when gaps in the specification were spotted.

In short, there did not seem to be an existing design notation which allowed designers to specify an environment in a way that could be interpreted by a programmer.

ENVISIONING A BETTER DESIGN DOCUMENT

With the analysis of the design document we were able to identify short-comings of the design document as a specification method. What was interesting, however, was they way in which the designers had used floor-plans in their discussions. The programming team had also used floor plans, but annotated in a different way. By forcing the designers to use floor-plans, they would implicitly be providing information about where objects are; what the objects would interact with and which objects are only part of the back-story (such objects would not appear on a floor-plan). But the question remained of whether designers would use floor-plans and if they could be used to capture any information the designer was interested in.

Upgrading Floor-plans

We were inspired by Fencott's theory of "perceptual opportunities" which he uses to construct VEs [6], Perceptual Opportunities model the content of a VE by describing the content in terms of psychological qualities which attempt to manipulate the player's attention through the player's perceptual system. In this way designers can construct a VE by considering how the player can be guided in a VE by the objects and properties of the VE. There are three types of perceptual opportunities: sureties, shocks and surprises. Objects which exhibit predictable behavior in a VE are called sureties and attempt to make the world believable to the user. An example of a surety is ambient sound – sound which, if correctly chosen could communicate the nature of the environment. Shocks are objects or properties of the world which are perceived by the user as unbelievable and are by-products of the construction of a VE. An example of a shock might be ambient sounds that suddenly stops or texture maps that do not tile correctly on a building. The perceptual opportunity

which we are particularly interested in is surprises. 'Attractors', 'connectors' and 'rewards' make up the three basic types of surprises. Attractors are ascribed to content which draw the user around the VE. Through the use of animation, color, sound and mysterious content, the designer can attract the attention of the user and thereby draw him to move around the VE space. Connectors are surprises which encourage the user to take a particular route of action in the VE. An example of a connector might be a bridge connecting the user to an attractor. Rewards are the content which make the user feel satisfied for their effort to follow the attractors and connectors. A reward might be something the user can "pick- up".

Fencott used perceptual maps to diagram the perceptual content of the VE. These maps consisted of tables showing the relationship between attractors, connectors and rewards. In like manner we desired to show interaction content (which implicitly implies perceptual content) but using a floor-plan map instead of a table in order to gain the benefits of using a floor-plan listed at the start of this section.

Refining the Diagrams

Essentially our research was now about creating a new visual formalism based on augmenting floor plans with Fencott's ideas. In order to create this formalism, we employed an iterative process and created a series of low fidelity prototypes which we would test by creating descriptions of existing environments and then passing them on to programmers (who were not directly involved with our team) to see how easily the diagrams could be converted into functioning environments. An early version can be seen in Figure 1. Over time, as our design began to stabilize, we worked inside PowerPoint, creating a custom palette of the symbols we needed.

Artifacts

These objects in Figure 1 included three characters (the barman, Natasha and Stopsign the bouncer) and props such as the jacket and sunglasses. By placing the objects spatially, one could see the competing attractors which would draw the player to various locations in the world. This would allow the designer to plan the attractors by deciding where the objects would go in the environment. By allowing the designer to place an object this way, it would also provide the programmer with authoring information, indicating the placement of objects inside the VE.

Interactions

Besides showing props and character objects, the designer would also need to document the actions which occur when the player attempts to interact with a character or object in the world. We created symbols for each of the actions that the authoring tools supported. For example, in Figure 2, the

 symbol is used to denote that an audio file is played

when the avatar enters the area (denoted by the circle) around Natasha.
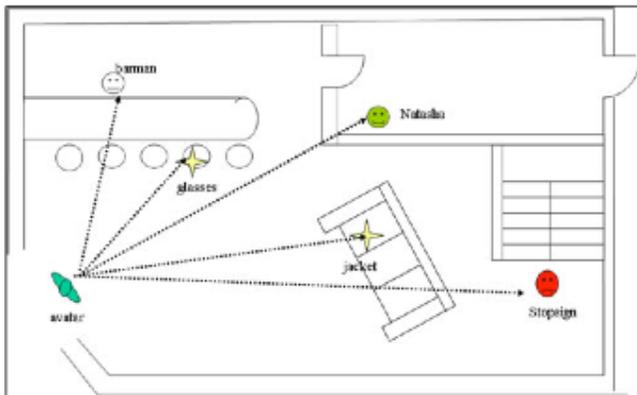
Figure 1 - objects placed in the world and attractor lines showing the competing attractors which the avatar must decide to follow
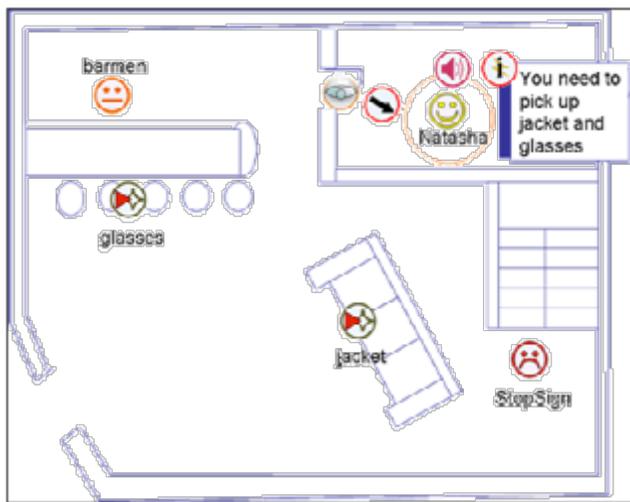
Figure 2 – Refined notation

Following the placement of characters and props, we considered what the designer might like to document in terms of narrative. Drawing on the design document's emphasis of the back-story, we needed some way to document how the designer would progress the story and thereby reveal it to the player. By using an icon to show where part of the plot is revealed to the player, the designer would have some way to document the story on the floor-plan. From an analysis of the design documents we created three types of narrative icons:

- Plot-points: used to show that part of the back-story is now revealed to the player.

- Plot-reversals: used to show if the player is distracted in some way from having the back-story revealed.

- Instructions: used to shows where instructions are provided to the player which help the player to progress to the next plot-point or accomplish some task.

We also characterized the characters and props. The characters can either be good (help the player to progress in the story), neutral (characters that merely bring believability and context to the game) and bad characters (which try to prevent the player from accomplishing his mission). A look at Figure 2 shows the narrative icons, props and characters. The player interacting with Natasha in this floor-plan shows

that Natasha, , a good-guy, is providing the player

with an instruction, , "you need to pick up jacket and glasses" and does this through an audio file.

Programming
It must be stated that the design language and tool is for the designer to specify interaction information and not the programmer. The programmer must be able to "read" the language and glean the information needed once the designer has finished planning. In our discussions with the programmers, they required information about the various types of 'triggers' that would invoke actions (object methods) within the environment. Our programmers identified the following: proximity triggers, tripwire triggers, user-selection events (mouse input and keyboard input), timer triggers and collision detection. We provided an icon for each of these triggers but drew the proximity trigger as a circle and the tripwire trigger as a line on the surface of the floor-plan. Figure 2 shows the player interacting with Natasha. The condition that calls the action is a proximity trigger drawn around Natasha with the circle. Once this condition is called, the audio file is played. The proximity sphere annotation (the event or condition type) together with the audio icon (the action) make up the interaction: "if user walks through proximity sphere, play audio file".

Rules for the placement of Icons and Annotations
We were not strict with defining rules for the icons and annotations as we wanted to see how the designers would use them and if their use would automatically create rules. We see this as a strength of the method as we aimed to facilitate a shared understanding within a group (we did not want to have them slavishly follow our rules) and create a shared meaning from the base we provide. With our use of the icons and annotations we found that the actions which belong to an object (like animation or spatial audio), must be located close to the object to show ownership. The proximity trigger must belong to some object and therefore is always around it. We provided the designers two types of text labels. The first is a text label which the designer can use to annotate something with text. The next type of label is the scene label. This label is intended to allow the designer to write a short sentence describing the interaction occurring on the current floor-plan.

By considering the designers and programmers needs, we were iteratively able to come up with a visual language with

which to describe interaction information. Thus we were ready to consider designing a software tool that would manifest our icons and annotations to the designer and allow the designer to easily create interactions in a visual way. A sample of icons is provided in Figure 3.

Icon and Name	Definition
Good Character	This represents a character in your world. This character icon means that the character aids the user to help him accomplish the task or find the purpose in the VE. E.g. Guides, characters that reveals information.
Neutral Character	This character can be interacted with but does not in anyway advance the avatar from one place to another or detract the avatar from accomplishing his task. This could be any character that is an accessory in the environment.
Bad Character	This character is used to deter the user from accomplishing the task in the VE. E.g. A character that can kill you.
Interactive Prop	This is a prop that the user can interactive with – or do something with – like pick up, move, open, view close up.
Static Prop	This is a prop that is significant in the world, but cannot be interacted with by the user. It is static.
Waypoints	A waypoint flag is used to show a point along a path. The flag and the lines connecting the flag make up the waypoint. Waypoints are used to show the path which non-player characters can follow.
Episode Portal	This icon is used to show that a portal exists at this point. A portal is a point of entry or exit. **This icon represents a portal between episodes**. How would the user get from one set to another? Usually there is a trigger the user crosses to get through to the next set in the next episode. In CAVEAT you must remember that you cannot traverse back to previous episodes. The user can only move forwards, advancing the episode map.
Set Portal	This icon is used to show that a portal exists at this point. A portal is a point of entry or exit. **This icon represents a portal between sets**. How would the user get from one set to another? Usually there is a trigger the user crosses to get to the new set. In CAVEAT you can design so that the user can traverse back to sets they have already been to within an episode.

Figure 3 – Sample of some Design Notation Icons

EVALUATING THE DESIGN NOTATION

In order to evaluate the design notation we again used media students. Fortunately a full year had passed since our first intervention and we were able to evaluate the notation with a similar body of students to that which took part in the original study. The key difference in the course this time was that the students did not have free reign in the use of design notations, but had to use our new notation. So, instead of teaching the students pseudo-code, use-case diagrams etc., they were given three lectures on our design notation and given the PowerPoint symbol palette to aid in constructing their final designs. As we realized it would not be possible to capture all the designers' ideas using our notation, they were required to submit a free-text 'screen play' document to augment the floor-plans.

Just as before, programmers from our research team were attached to each group of students to observe the discussions within the team and also provide technical input on the designs. In an effort to triangulate the ethnographic discussions within a group, we also had the students conduct an artifact walkthrough [11] of the final diagram to ensure that our interpretation of the diagrams matched the ideas the designers were wanting to express. We used this technique by conducting unstructured interviews where each designer was asked to explain their design and to describe the use of the icons and annotations they employed. These interviews were individual, as apposed to focus group discussions, ensuring that the designer's responses were not influenced or overpowered by the other designers.

Results

As we had hoped, the designers could indeed use the floor-plans intuitively. Designers also seemed happy to incorporate non-spatial information (such as plot points) onto the floor-plans. By explicitly placing characters and props on the floor-plan, many of the ambiguities experienced by the programmers disappeared. When implementing the environments described by this notation, it was also much easier to understand interactions as the trigger points were all clearly marked. For example, in one environment we implemented only one of the nine triggers used was not properly specified. In terms of specifying code that goes with the game logic, there were still a few problems in understanding the designer's intent. However, the programmer did not have to aggregate information about the interaction as with the previous notation, and ambiguities could be rapidly cleared-up by consulting the designer. However, we did discover some problems and shortcomings in the notation.

Interactions

We found that the designers made use of the screenplay document and the floor-plans to record different types of information. While the floor-plans showed the game design in an abstract way, along with the required interactions, the screenplay documented the story. The screenplay was not a document intended to be used by the programmer. However, we found that the screenplay not only documented the story, it also revealed rules of the game that were not explicit in the floor-plans. In one example, the designer had drawn a trigger circle around a character, but specified the interaction in the screenplay. Without reading the screenplay, the programmer would not know have known what was to happen. We had meant for the designers to use the label function to record rules in the floor-plan. These labels were not often used and when they were used there was not enough specified to identify the global rules of the game. Thus the floor-plan language labeling did not encourage the designers to record game rules while designing.

We found that during the artifact walkthroughs, the students explained more of the storyline verbally than was recorded in both the screenplay and the floor-plans. While explaining their game during the artifact walkthroughs, they often pointed at each icon and annotation to explain the scene. The icons served as pointers to remind the designers what they needed to share about their game. It is interesting to

note that this information was also not specified in the screenplay document and so the plot point icon was useful for the designers to deliberately show in which scenes the players should discover the plot. Thus the plot points became useful placeholders in communicating verbally about the story of the game.

Icon Design

In our design of the visual language, we hoped to provide the designers with a complete set of icons and annotations and a standard way for expressing those icons and annotations for a VE design. The experiment helped us to reveal the language discrepancies and identify areas where improvements can be made.

There were two problems with the use of the prop icons. One of which was that the prop icons were not re-sizeable. The size of the icon represented a physical model and therefore the designers needed some way of showing the scale of an icon in relation to other icons. The other problem was to do with the static prop icon only. The designers were not sure how many static prop models they should represent on the floor-plan. We found that some static prop icons were drawn using the marker tool and even some of the significant static props were left out.

Waypoints

The designers creatively used icons and pencil line drawings to show waypoint information. In one case the waypoint was drawn with a pencil tool showing the path and an arrow at the end of the line to show the direction. To mark the start position of the waypoint a flag was used. Another diagram showed the waypoints path using flags to mark the path and then used attractor arrows to show the direction of movement. Although both examples managed to show all the waypoint information necessary, the language which we provided the designers for waypoints was not standardized. We intended that the waypoint flags would be enough to show the waypoint path. We had not standardized the starting position, the direction of the prop which moves along the waypoint, the animation which the prop does along the waypoint, nor did we specify that a prop was attached to a waypoint.

Attributes

We also did not explicitly provide a way for the designer to document the avatar or character attributes. This is usually called the character's inventory, a concept borrowed from gaming. Having an inventory translates to having a set of variables describing the items that are placed in the inventory and rules which can be applied to those variables. Every team made use of an inventory. There was no official way that the designers could specify the variables and rules they required. Some of the inventory variables were implied by the use of the interactive props which the player at times could collect. However, the items which did not have a physical counterpart in the environment, for example "health", were not diagrammed on the floor-plan. Further, the rules regarding the inventory items were not diagrammed on the floor-plan. Even with those inventory

items shown on the floor-plan as interactive props, it was difficult to know whether the item was in the inventory (and therefore not seen in the environment) or out of the inventory (and therefore seen in the environment). One designer suggested in the interview that in order to show that a player had collected an item already, one should be able to attach the interactive inventory icons to the avatar icon. Our floor-plan language needs to be improved with regards to recording a very clear specification of variable items and their rules.

FREND

In parallel with the efforts to create the design notation, a systems programmer was creating a software tool to embody the final notation that resulted from the design process. Two weeks after submitting their final PowerPoint design, we recalled the designers to evaluate our new VE design tool called FREND – see Figure 4.
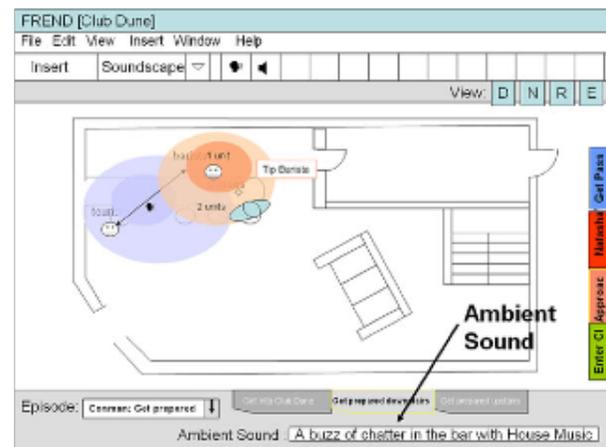

Figure 4 – The FREND interface

FREND was designed to follow the look of the PowerPoint interface that we had been prototyping. At the level of the floor-plan it is almost identical to the PowerPoint interface. However, it adds other features, to break complex VEs into part-whole relations, to manage assets, link to design documents and to link to an authoring tool and load virtual environments that are beyond the scope of this study.

This evaluation of FREND required the subjects to re-create their design using only the screenplay as reference. Primarily we were interested to see how intuitive the notation had been, so subjects were not re-taught the meaning of any of the icons nor given any form of reference sheet. Our evaluation lay in comparing the original PowerPoint submission with the newly created FREND design; our hypothesis being that consistency would be a good measure of the appropriateness of the design notation.

Procedure

Each group of four students was divided into pairs. Each pair was then asked to recreate their design using FREND. Pairs were chosen as we wished to exploit constructive interaction in order to understand how the students were

rationalizing about the recreation of their designs. It was felt that constructive interaction with both members being subjects would give us the most un-biased insight – by working in teams of two instead of individually, we hoped to pin-point problems as the students discussed the interface and floor-plan language verbally to each other. Such information could be missed using an individual user-testing as it is unlikely the designer would communicate their software difficulties sitting alone.

Each computer was equipped with a video recording device which recorded the visuals and the audio of the students working with FREND. The screenplay document of each group was given to the designers during the experiment as a means of providing consistency between pairs of the same group using FREND and to jog the memory of the designers.

Results
Overall, we were surprised at the consistency between the original submissions and those created in FREND. In a few instances the placing of props was different and the choice of trigger type had changed. However, these differences are not due to whether or not the notation was learnable but rather a choice by the designers.

Outcomes
Encouraged by the seeming robustness of our design notation, we set about improving FREND to accommodate the concerns that arose from analyzing the PowerPoint design submissions.

Icons
Whilst it is possible to implement re-sizable icons to represent props, there is little we could do about forcing the designer to draw props with a "props" tool rather than using the pencil tool. We would have had to remove the pencil tool from the toolset in order to force the designer to use the prop tool, but this would have compromised overall flexibility too much. It could be that some day we figure out every interaction and object possible and get rid of the general-purpose pencil tool, but until then, it was felt best to leave it intact.

Attributes
Firstly, we allowed the designers to assign attributes and objects to an avatar. By double clicking on the avatar, a form appeared which allowed designers to specify attributes (by giving a name, default value and a text box allowing the specification of rules relating to that attribute) and any objects that character may have in their inventory. This functionality would not have been possible in PowerPoint.

Waypoints
The original floor-plan language simply allowed for flags to show waypoints and then designers joined the points to show the completed path. Now waypoint flags are still provided in an automated way but an additional bigger flag marks the starting position of the waypoint. On defining a path, the designer drags in the big waypoint flag, places it on the floor-plan, after which a pop-up menu form allows

the designer to select the object to attach to the waypoint and define what animation the object should play along the waypoint. Each waypoint defined has a system generated number so that one can identify the specific waypoint the character is attached to. After placing the big waypoint flag and attaching the waypoint to the character, a waypoint drawing tool is then used to draw the path. Once the designer is finished drawing the path, the system automatically places the arrow in the direction the path was drawn and places mini-waypoints flags along the path. This new way of specifying waypoints, shows the waypoints path, direction, the object which is attached to it and the animation which the object must perform along the path. The notion of time and synchronization is not inherent in the waypoint notion – if users require time information, then a 'timer' needs to be added to the path.

Final Outcome
All but one of the subjects who took part in the evaluation of FREND preferred it over PowerPoint. Ultimately FREND was able to manage large VE's as systems of episodes and scenes. The icons could link to asset libraries where complete description of the objects could be built up. FREND also generated code stubs which could be used by the other VR authoring tools in the suite we created. Overall the system provided a very direct way of moving from the designer's ideas to a working environment.

CONCLUSION
The wider goal of this research effort was to broaden the applicability of virtual reality technology. In the work reported here, we tackled the problem of communication between the design teams that create virtual environments and the programmers who implement them. From our review of the literature in this field, this seemed to be a common problem, but there did not seem to be any general solution. Borrowing from a variety of disciplines, we taught designers a broad range of notations in the hopes that some of these could be easily translated by programmers. When this approach failed, we created our own notation based on Fencott's work [6,7]. After developing this notation into a set of symbols of for use in PowerPoint, we were able to train designers to create documents that were meaningful to them and which could be translated into code with much less effort than was possible with existing techniques. The notation was further refined and them embodied in a software tool called FREND.

We believe that we have made significant progress in our goal of creating a design notation for virtual environments that is intuitive to designers and provides useful information to programmers. The amount of information required from the designers by the programmers was greatly reduced between the first group of students and the second group. By forcing the placement of icons in physical space, and providing a symbol set that mapped directly to the authoring tool's functionality, we were able to greatly reduce the amount of time required to create an working environment. Furthermore, the latest version of FREND can

be used to automate some of the process and generate code stubs that incorporate the designers' ideas as comments within the stubs.

From the designer's perspective we were able to create a tool that captured most of the concepts they were interested in expressing about the layout and interaction with the environment. They found the notation to be intuitive and easily remembered.

Whilst we believe our floor-plan notation is a worthwhile approach to this problem, there are some inherent limitations. Primary among these is the fact that we are representing a three-dimensional space in a two-dimensional diagram. Whilst there are conventions to do this (e.g. using plan and side views) most of the designers we worked with tended to conceptualize their designs in two dimensions, limiting the need to provide 3-dimensional visualizations. Another weakness in our overall approach is the involvement of programmers as intrinsic in the process. Ideally we would like to create a tool for non-experts which would let them create these environments on their own. Whilst this is possible to some degree with existing tools, we did not find them to be sufficiently general-purpose for our requirements. Finally, we accept that the notation is most appropriate for narrative-driven game environments and would not be appropriate in the creation of every type of virtual environment. It is our hope that by exploring new notations, such as the one presented here, that we open up new research areas to enable the next generation of VE authoring tools.

ACKNOWLEDGMENTS

REFERENCES

1. Brown, S., Nunez, D. and Blake, E. Using virtual reality to provide nutritional support to HIV+ women. In Slater, Mel, Eds. *Proceedings: The 8th International Workshop on Presence — Presence 2005*, (2005) 319-326.

2. Pausch, R., Burnette, T., Capeheart, A., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, White, J. IEEE Computer Graphics and Applications, May (1995), 8–11.

3. Conway M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K., Deline, R., Durbin, J., Gossweiler, R., Koga, S., Long, C., Mallory, B., Miale, S., Monkaitis, K., Pattern, J., Pierce, J., Shochet, J., Staack, D., Stearns, B., Stoakley, R., Sturgill, C., Viega, J., White, J., Williams, G. and Pausch, R. Alice: lessons learned from building a 3D system for novice users. *Proceedings of the SIGCHI conference on human factors in computing systems.* The Hague, Netherlands, ACM Press. (2000), 486-493.

4. Kaur, K., Sutcliffe, A. and Maiden, N. A design advice tool presenting usability guidance for virtual environments. *Proceedings of Workshop on User-Centered Design and Implementation of Virtual Environments.* (1999)

5. Schwartz, P., Bricker, L., Campbell, B., Furness, T., Inkpen, K., Matheson, L., Nakamura, N., Shen, L.-S.,Tanney, S., and Yeh, S. Virtual *Playground: Architectures for a Shared Virtual World. Proceedings of ACM Symposium on Virtual Reality Software and Technology* (1998) 43-50

6. Fencott, C. Towards a design methodology for virtual environments. *Proceedings of Workshop on User-Centered Design and Implementation of Virtual Environments. England, University of York.* (1999) 91–98.

7. Fencott, C. Virtual storytelling as narrative potential: towards an ecology of narrative. ICVS 2001, LNCS 2197, (2001) 90–99.

8. Tanriverdi, V and Jacob, R.J.K. VRID: A design model and methodology for developing virtual reality interfaces. *VRST '01: Proceedings of the ACM symposium on Virtual Reality Software and Technology held in Banff, Canada.* ACM Press, (2001) 175-182.

9. Jacob, R.J.K., Deligiannidis, L. and Morrison, S. A software model and specification language for Non-Wimp user interfaces. *ACM Transactions on Computer-Human Interaction*, 6, (1999) 1-46.

10. Cho, Y., Park, K.S., Moher, T. and Johnson, A.E. Mediating collaborative design for constructing educational virtual reality environments: a case study. *Lecture notes in computer science,* 3190, (2004), 30 – 37.

11. Muller, M.J. Catalogue of Scenario-Based Methods and Methodologies. *Report 99-06.* (1999) Massachusetts, IBM Watson Research Center.