Learning from Games: HCI Design Innovations in Entertainment Software

Jeff Dyck[1], David Pinelle[1], Barry Brown[2], and Carl Gutwin[1]

[1]HCI Lab, Deptartment of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada, S7N 5A9
jeff.dyck@usask.ca; http://hci.usask.ca

[2]Department of Computer Science
University of Glasgow
Glasgow, Scotland
barry@dcs.gla.ac.uk

ABSTRACT

Computer games are one of the most successful application domains in the history of interactive systems. This success has come despite the fact that games were 'separated at birth' from most of the accepted paradigms for designing usable interactive software. It is now apparent that this separate and less-constrained environment has allowed for much design creativity and many innovations that make game interfaces highly usable. We analyzed several current game interfaces looking for ideas that could be applied more widely to general UIs. In this paper we present four of these: effortless community, learning by watching, deep customizability, and fluid system-human interaction. These ideas have arisen in games because of their focus on user performance and user satisfaction, and we believe that they can help to improve the usability of other types of applications.

Keywords

Computer games, game interfaces, user communities, interface customization, interface design

INTRODUCTION

Computer games are an enormously popular and successful type of interactive software. This success has occurred even though game interfaces and interaction paradigms are very different from those of other applications. Because of their focus on system performance over consistency, games have nearly always ignored the windowing systems, the standard widget libraries, and the toolkits that define the look and feel of conventional systems. In this way, game UIs were 'separated at birth' from their siblings, and grew up in a very different design environment.

In particular, this environment does not place restrictions on how things must look or how interaction must be carried out with the user, but it does strongly reward innovation and performance. The driving forces in game design are user performance, satisfaction, and novelty: gamers have come to expect new, cool features that they have never seen before, features that help them play in more efficient and more interesting ways. As a result, games have both become early adopters of new HCI technologies as well as innovators in the area of HCI interaction design.

Examples of early adoption are many, and include transparent overlays in Diablo II (studied in [2]), transparent menus in Everquest ([9]), radar views in Warcraft ([8]), gestural commands in Black and White ([18]), speed-coupled flying in Grand Theft Auto ([16]), and radial menus in Neverwinter Nights ([11]). However, games do not just adopt; the competitiveness of the market and the expectations of the player communities lead game designers to produce both variations on old techniques as well as completely new ones. This paper is about the innovations that have grown up entirely in the game world – techniques and approaches that can now help to advance the design and usability of conventional applications.

HCI researchers have considered games before: in the early 1980s, Tom Malone looked at what makes games compelling and how these properties could be applied to applications [14]. In the ensuing 20 years, however, games have evolved enormously, but their progress has gone largely unnoticed. A second look at the design and interaction innovations – this time in modern games – was long overdue.

We have taken this look by carrying out a design review of fourteen state of the art PC games from several genres. Our goal was to identify novel contributions that provide clear benefits to users in game domains, contributions that could be also be employed to help improve usability in conventional applications. In this paper, we introduce four of these innovations:

- *effortless community* – games make it easy to form, join, and participate in communities of users;
- *learning by watching* – games help people learn the application by watching 'over the shoulder' of more experienced users as they work;
- *deep customizability* – games give users the power to modify and extend any aspect of the UI, and allow them to share those modifications with others;
- *fluid system-human interaction* – games communicate information to users in ways that do not demand the user's attention and do not interrupt the flow of work.

Even though games are often seen as being "just for kids" or "just for entertainment," games have had to address many of the same interaction and interface issues that affect more conventional systems. The design ideas that we present below are eminently applicable to everyday situations with regular software; and we suggest that reuniting the separated siblings can have distinct advantages for both software usability and HCI research.

METHODOLOGY

We examined fourteen recently released, commercially successful games (see Table 1). We looked at games from several of the main game genres, selecting titles that have not only been successful in the marketplace but that have also been highly commended with reviews and awards.

Game	Genre	Play
Warcraft III	Strategy	S, M
Ghost Recon	1st-person shooter, strategy	S, M
Rogue Spear	1st-person shooter, strategy	S, M
Half-Life	1st-person shooter	S, M
FIFA World Cup	Sports	S, M
Medal of Honor	1st-person shooter	S, M
EverQuest	Role playing	M
Diablo II	Action, role playing	S, M
The Sims	Simulation, strategy	S
Neverwinter Nights	Role playing	S, M
Comanche 4	Simulation	S, M
MechWarrior 4	Action, strategy	S, M
Grand Theft Auto	Action	S
Black and White	Strategy	S, M

Table 1. Games studied. S=Single player, M=Multiplayer.

The games were explored with a variety of methods. First, we played the games, both as individuals and groups, and kept diaries of our game playing experiences[1]. Second, we held group analysis sessions for each game, where we catalogued its interaction techniques, critiqued its interface, and looked at each main element of the game's design. Third, we observed (in person) how players other than ourselves used the interfaces, and watched (on-line) the text conversations of groups playing on-line games. Finally, we collected game reviews and discussions from review sites.

From these activities, we produced a list of game design elements and approaches that are novel and that could be useful in conventional applications: effortless community, learning by watching, deep customizability, and fluid system-human interaction. In the next sections, we will discuss what each of these innovations are, how they work in different game situations, and how they can be applied to the interfaces of conventional systems.

EFFORTLESS COMMUNITY

Games make it easy to participate in online user communities and easy to form groups within them. User communities are extremely valuable resources that help people resolve problems and provide collaborators for new projects. However, these interactions require a critical mass of users who are available on-line [17], and also require that people be able to find (or form) the right subgroups for their specific interests and needs [6, 3]. Because games are intrinsically interested in multi-user interaction, they have had to address these requirements, and have become very successful at meeting them. They make it easy to obtain critical mass; they make it trivial for users to connect to the community; and they make it easy for users to form and find subgroups within the community. Even though conventional applications come from a single-user mindset, using the techniques that games have developed can help them to make better use of the natural community of users.

Getting to critical mass: the natural community

The users of any application form a natural community with an obvious common interest. The graphic designers who use Photoshop form a natural community, as do the Java programmers using JBuilder, and the architects who use AutoCAD. These communities get together – if they do at all – on line, most often in newsgroups and websites. The communities can also be large: for example, the comp.graphics.apps.photoshop group contains more than 140,000 discussion threads. For many applications, there are enough concurrent users worldwide to form a natural community that is large enough for people to find answers to questions, to comment on content, or to find collaborators for an upcoming task.

However, conventional applications do not make direct use of this natural community – users are disconnected from one another and unaware of others who are using the system. When communities do exist, participation occurs outside of the application and asynchronously (as in newsgroups). Users who want to participate in communities must use third party tools to find and communicate with each other. Not only does this require extra effort, but it also takes discussion out of the context of the application, which can make communication more difficult (e.g. describing the specifics of an interface problem).

In contrast, multiplayer games have successfully integrated the natural community with the applications themselves. This integration guarantees that there is a critical mass of users (assuming that there are enough users in total) that participate in the same location and that are available for collaboration. Games do two things in particular that enable and manage community involvement: they make it trivial to connect to the online user community, and they make it easy to locate collaborators and form subgroups.

Effortless connection to community

Games provide simple and direct access to their online communities through the application itself. Usually, games require only a single click and a login for a user to connect to others. Making a connection to the community, of course, is a requirement for multi-player games, and the simplicity of that connection process has arisen from the frequency of the task. However, even though CSCW researchers know about the problem of getting connected

[1] Despite the arduous nature of the research, there was reasonable enthusiasm for this phase of the work.

(e.g. [Greenberg & Cockburn]), it has never been adequately solved outside of games.

Games make the connection process simple by ensuring that dedicated, reliable game servers are always available as contact points for the user community. Some games also allow users to host communities on their own game servers, which similarly requires just a few clicks from within the application itself. By making participation easy, games successfully attract large numbers of users to their online communities. Everquest currently has the largest user community of any game, with as many as 100,000 people connected at once; and when online, these people interact not only as players in the game, but also users of the Everquest application (see figure 2).

Effortless connection solves the problems of getting to the community and achieving critical mass; but once users are online with a large group, the problem is not finding people, but finding the right people – those with common interests and particular knowledge.

Identifying and forming groups with collaborators

Games make it easy to locate and form groups with potential collaborators from within a large community of online users. Many games have thousands of concurrent online users at any time, but people usually have different requirements about who they wish to collaborate with depending on their current tasks. For example, a general question about a workaround for a system bug might be directed to a wide group or to anyone who is nearby; but in most situations people need to find collaborators with (e.g.) compatible personalities, similar levels of expertise, and common interests. Games provide rich support for finding and forming groups with collaborators. There are two distinct approaches: meeting places, and in-game grouping.

Meeting places are portals where people can get to know each other, look over potential collaborators using statistics and stored profiles, discuss strategy, and solve technical problems with the game. These are generally used for games that are oriented around small group play and limited-time interactions. For example, Battle.net is a network meeting place integrated into Warcraft III that provides discussion forums and player statistics, allows people to create custom games and advertise for players, lets people join custom games, and provides an automated matchmaking service that groups compatible players (see figure 1).

The second approach to group formation is to have people form groups through the interaction mechanisms of the game itself. This approach is used for massively multiplayer games in which thousands of people play in the same game world and where there is no defined end to the game. The on-going nature of these games allows people to build up an identity within the world, and these identities help people to find appropriate partners:

- guilds: users can form or join guilds which have specific purposes and exclusive membership criteria
- location: the spatial nature of the game provides a natural grouping mechanism, since people in a particular place likely have something in common;
- conversation channels: games allow people to create and join chat channels with specific purposes (such as discussing how to reduce network lag);
- friend lists: games provide 'friend lists' for easy access to particular groups, and 'block lists' to exclude others;
- explicit teams: games allow the creation of explicit groups of up to six people (which restricts communication and enables tracking mechanisms) for carrying out tightly coupled tasks in the world;
- visual identity: games with avatars enable users to show their skills, loyalties, and expertise in a visual form through the appearance of the avatar, giving others an easy way to assess potential collaborators.

Figure 1: Battle.net, the meeting place for Warcraft III players. Players can chat, view each others' rankings, build custom games, join games, or use a matchmaking service to automatically find collaborators based on preferences.

Using natural community in conventional applications

Connecting the natural communities within conventional applications is a radical concept – since most applications do not think of themselves as 'groupware' – but it is one with tremendous potential. Although game players have additional motivations for being online, gamers and workers share many reasons for collaborating. All users need to build expertise, have questions answered, discuss approaches, and solicit feedback. These tasks can all be performed through collaboration with the user community, if that community is easily accessible and can be partitioned appropriately. Games have not only been very successful in building communities and grouping people with common interests, they have shown the value of real-time online communities in getting tasks done efficiently. Consider the potential of the natural community in an application like Photoshop. With the number of concurrent users, it would be easy to build a large real-time

community with a population in the tens or hundreds of thousands. With integrated community mechanisms inside the application, Photoshop users could get answers to questions, find out about new features, discuss problems, and (as discussed further below) learn from watching others at work.

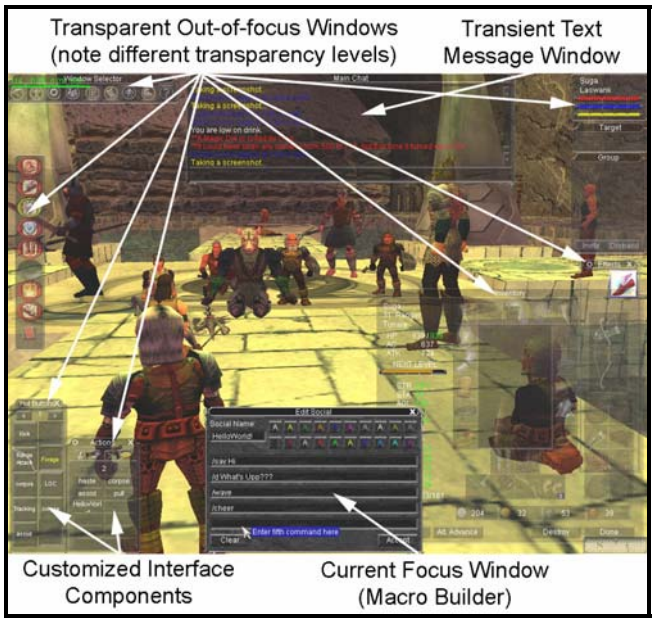

Figure 2: Everquest can have in excess of concurrent 100,000 users. This figure shows 2 palettes of customized interface components, the macro builder (see figure 3), and a scrolling message window. Out of focus components are more transparent and the in-focus macro builder is less transparent.

LEARNING BY WATCHING

One aspect of community that games support very well is the idea of learning from other more experienced users. In communities, individuals regularly learn by observing others. However, when those individuals are distributed and do not see each other face-to-face, this type of observational learning can be difficult. Computer applications have the opportunity to provide users with support for observing remote users so that they can benefit from their expertise. However, most applications do not provide a straightforward way in which this type of observation can take place. Even in groupware applications, the support for embodiment and workspace awareness may not be adequate enough to allow users to easily understand the actions of others at a fine-grained level [10].

In contrast to conventional applications, many multi-player computer games provide strong support for learning by allowing users to observe others who are present in the shared game world. This learning is usually made possible through the use of real-time awareness and embodiment information that provides each user with a detailed understanding of the specific actions that are being carried out by others. For example, GhostRecon represents each user using a 3D avatar. The avatar can crouch, crawl, jump, run, open doors, and pick up items; combinations of these and other activities are all observable, and by observing this a novice can learn from an expert. Support for this type of observational learning in this case is tied to the use of an avatar; however, it is the conveyance of the embodiment, awareness, and task based information that allows for this type of learning to occur, regardless of the mode of conveying the information. Therefore, observational learning can reasonably be transferred to other applications by allowing remotely located users to observe the actions of others, and by providing rich embodiment and awareness information that allows for the easy interpretation of fine grained actions.

While shared views of workspaces and activity awareness have been explored in groupware literature [7], this type of learning has never been considered for applications that are traditionally considered as single-user systems. The success of desktop sharing applications such as VNC suggests that there is a need for this type of observational learning, both at the operating system level and at the application level. However, since these desktop sharing applications are general tools, they do not always provide adequate information for others to interpret task-specific activities that are carried out in specific applications. To adequately support observational learning, games show us that a fine grained understanding of action sequences is required. Taken a step further, desktop applications can convey awareness of interactions with the workspace (i.e. mouse pointers, drop-down menu selections), but also information about the specific interaction events that are used to trigger events at the application level. For example, information about specific key sequences used by the expert user can be displayed to the observer in order to allow them to learn common shortcuts and keyboard-based command sequences.

DEEP CUSTOMIZABILITY

Games consider modifying and extending the UI to be a commonplace and necessary part of using the system. They give users a set of simple but powerful mechanisms for changing the UI to better support particular styles, tasks, and situations. Although most conventional applications offer some type of customizability, their facilities are limited and require considerable effort to use. Games, in contrast, take an extreme view of interface customization: in many games, the form and content of the UI is almost completely under the user's control. We found three innovative customization mechanisms in games that could be applied to conventional applications: anything-goes UI malleability, natural extensibility, and portable customizations.

Anything-goes interface malleability

Game interfaces are plastic; they are designed to be changed. Gamers have learned that different interface configurations can greatly affect performance in different

game situations, and that no single configuration can be appropriate for all tasks. This is equally true of complex conventional applications like Word or Photoshop; the difference is that gamers see the extra effort required for a suboptimal interface configuration as the difference between victory and defeat, or life and death.

The malleability of game interfaces can be seen in two areas: interface layout, and mappings from controls to functions.

Everquest is a good example of layout malleability. There are many functions in the system, and different ones are more or less useful in different scenarios (e.g. attacking, defending, exploring, or buying and selling). As a result, UI elements in Everquest have been designed to be picked up and moved or copied. When a user holds the mouse button down over an element for a longer-than-normal time, the element detaches from its base and sticks to the mouse cursor. The user can then put it down in a new location anywhere on the interface. In addition, the game makes it simple to create a new container for commands, so that a custom palette of tools for a particular purpose can be set up and located ready to hand in seconds, and with only a few mouse clicks (see figures 2 and 3).

Game players use this capability all the time – and an indication of its simplicity is that they use it not just to satisfy long-term preferences, but also to address short-term situations that may last only a few minutes.

The second type of malleability involves the ability to remap the functions of UI controls. This practice arose from the need to set the functions of input devices (joysticks, mice, command keys), but has since been adopted for visual controls as well.

Users can both change and add to the functions that controls execute, and the latter is particularly common. For example, some players in perspective shooter games remap the 'move left' and 'move right' keys to add a 'crouch down' function. This type of remapping begins to look like a macro capability, which is discussed further in the next section. Having complete power over remapping may sometimes lead to chaos (e.g. remapping well-known buttons or letters on the keyboard) but undo and reset functionality allows users to play with the plasticity of the system, and encourages them to try many new configurations to see which if any will provide a benefit.

Natural extensibility

In addition to modifying the interface, games reduce the threshold of effort needed to extending the UI to the point where extensions become a natural and common part of the user experience. Extending a system's capabilities is a powerful concept that has been around since early editors like Emacs; however, in most conventional applications, extensions are difficult to build and difficult to use [13].

Macros in games are easy to build, and once built, are put into the interface as a normal command. An exemplar of natural extension capabilities is Everquest. The game comes with a number of 'button blanks' that act as containers for command extensions and macros. Within two mouse clicks, the user can be recording the actions that they want stored in the new button; and once they are finished, the new command is already part of the interface and ready to test or use.

Figure 3: Everquest's easy to use macro builder (right), palettes of customized components (left), and a component stuck to the mouse pointer being dragged to a new location.

In contrast, even though MS Word has powerful macro capabilities, creating a macro requires seven actions before starting, some of which are counter-intuitive (e.g. pressing a button marked "Close" to start recording). Five more actions are then needed place the macro onto a toolbar for use.

As with its layout-modification capabilities, the effectiveness of the Everquest macro capability is evident in its popularity: users define new commands as a matter of course, and do so even for a few minutes' worth of activity.

Portable customizations

Games have implemented their customization capabilities in a way that allows modifications and extensions to be saved, moved, and shared with others. This portability allows for an entirely new approach to customization, one in which users can have powerful situation-specific interface configurations, without having to do any work at all to build them.

In many games, macros, scripts, and layouts can be saved as ordinary XML files. This means that customizations can be edited and changed outside of the system – but more importantly, it means that they are portable and can be posted and traded within the user community. For example, within three months of XML functionality being added to Everquest, there were dozens of web sites with hundreds of modifications available for download. In addition, several 'mod kits' have appeared that greatly simplify the creation, editing, and installation of extensions, layouts, and skins. Users with little or no experience in the game can now use (and improve if

necessary) interfaces that have been built by experts and proven through hundreds of hours of use.

How much power to give the user in terms of customizing an interface has long been an issue in HCI. On the one hand, the argument goes, we should build the interface well in the first place, rather than depend on the user to fix the designer's usability errors (and it is likely that one reason that customizations became popular in gaming is that earlier interfaces were not very well designed). On the other hand however, complex applications that support more and more tasks will always run into the problem that no one interface setup will be optimal for any specific task [19]. Games' focus on performance and productivity has led them to favour the second of these two hands, and it is likely that for conventional applications that support many functions and expert users, it will be a valuable approach as well.

FLUID SYSTEM-HUMAN INTERACTION

Games deliver information in ways that minimize disruption to the user's work flow. The push for user performance in games has led the games industry to develop innovative communication strategies that demand less user attention and less user effort. Games use three novel approaches that result in a more fluid workflow: calm messaging, attention-aware interface elements, and context-aware view behaviours.

Calm messaging

Games deliver messages to the user in an unobtrusive way that does not require users to dismiss, acknowledge, or address them. In contrast, the approach taken by other applications often interrupts workflow: for example, notifications in modal dialog boxes demand attention and must be explicitly acknowledged. Although there are times when this can be justified, conventional applications often use these heavyweight mechanisms for all system-human communication. Games have shown that reducing demands on the user's attention can aid performance; through the use of sound, speech, transient text, and animation, games communicate in a calm manner that promotes a fluid, uninterrupted workflow.

Audio. Audio is an effective way to convey information to the user without adding visual clutter or breaking workflow [5], and games do this by using recorded voice notifications and by using spatialized environmental sound. Many games have libraries that contain thousands of high quality voice recordings and symbolic sounds that are used to communicate events. In Comanche 4, for example, recorded voice messages are used to convey information while the user controls the game's virtual helicopter. This approach is necessary, since the game requires timely responses from the user, and any delay introduced by the application would be unacceptable. This approach shows the potential for audio to inform the user of background information, with relatively little interference with the current task.

Transient text. Games make extensive use of transient text messages, which are automatically dismissed by the application and do not require any effort from the user. Some games display text for a predetermined period of time, and then it gradually fades from the user's view (e.g. MechWarrior 4). Others provide a message area that scrolls older messages out of view as new messages appear (e.g. Neverwinter Nights). These transient text techniques promote fluid interaction since action is not required to acknowledge or dismiss messages.

Animation. In games, subtle animation is commonly used to draw the user's eyes to a screen location that is associated with an event and to convey other additional information about the event. These animations are carefully crafted to match their level of visibility with the importance of the message that they are delivering. The ease with which animation is able to indicate direction, location, and priority allows messages to be conveyed very efficiently without interrupting workflow.

Most games combine a variety of these calm messaging techniques. For example, figure 4 shows how Warcraft III uses animation, transient text, and audio to deliver information to the user.

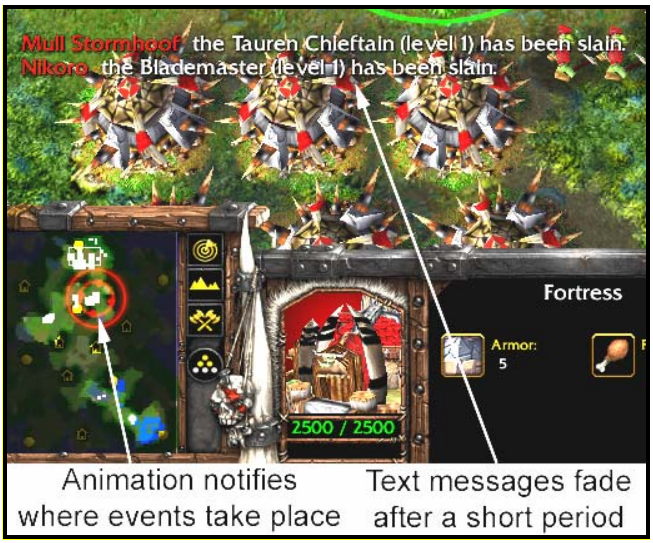

Figure 4: Calm messaging in Warcraft III. Text messages fade after a short time. Animated red concentric circles and arrows show the user where an event is taking place and recorded voice is used to communicate the type of event.

Some applications could benefit greatly from the fluid workflow that calm messaging provides; however, the addition of these techniques to other applications must be carefully considered. Game players do not typically have more than one active window open at once, so it is a reasonable assumption that they will see transient messages and animation that is generated by a game, whereas application users commonly have multiple windows open concurrently. Application users often don't pay attention to the applications that they have running (i.e. all windows are not necessary maximized and viewable), so messages could

be missed. Therefore, calm messaging may not be appropriate for delivering anything critical or urgent. However, for non-critical messages, calm messaging has the potential to improve user performance.

Attention-aware interface elements

Games use user interface elements that automatically modify themselves based on the amount of attention users are paying to them. This technique is effective at reducing visual clutter in areas of non-interest and increases the size of the useable workspace.

A new windowing system that was recently released for Everquest provides a particularly innovative example of attention-aware components. Each window has two user definable settings for transparency; one for when the window is in focus, and one for when the window is not. This allows the user to define the relative level of interest for each of the components. When the user enters the window area with their mouse pointer, the window automatically adjusts its transparency level to accommodate the increased interest in that area. A lower level of awareness of other more transparent windows is still maintained while not occluding the view of the game world (figure 2).

Context-aware view behaviours

Games automatically zoom, pan, and rotate the view of the workspace to best suit the task at hand. This reduces both the amount of effort required from the user to navigate and adjust the view of the workspace. For example, Neverwinter Nights allows the user to choose between three camera behaviours, which each automatically modify the view in a different way. Each of these camera behaviours is suitable for different types of tasks. Users can (and do) quickly toggle between behaviours using keyboard shortcuts to select the best behaviour that minimizes the amount of work they have to perform to navigate and adjust their view. Mastering the use of view behaviours greatly improves the playability of the game, and the importance of these behaviours is noted by their placement in the interactive tutorial, which teaches users how camera behaviours work as one of the first lessons.

This technique may not initially seem to be applicable to a 2D application. However, consider its potential use in a drawing application like Photoshop. Imagine adding a view behaviour that automatically scrolls when the user nears the edge of the screen and another that automatically zooms to keep the entire workspace in view regardless of what elements are added or removed. If the user could quickly toggle between these context-aware views, they could use the behaviour that automatically scrolls for navigating and performing touch ups in a detailed area without using the scrollbars, and the behaviour that automatically zooms could be used to build prototypes without having to adjust the zoom control to keep the whole area in view.

Fluid system-human interaction in applications

These techniques all have the potential to improve conventional application performance. Replacing non-critical message dialogs with calm messages would reduce the number of targeting tasks for the user and would not steal focus from their active window. Making interface elements attention-aware would result in fewer actions like resizing, opening, closing, minimizing, and maximizing windows. Context-aware view behaviours would reduce the amount of effort required to modify views and navigate the workspace. Applying these techniques to operating systems and applications would undoubtedly result in improved user performance in some situations.

DISCUSSION

In this paper, we identified novel design approaches in games and suggested how these innovations might be applied to other classes of software. Our success in finding transferable techniques reflects two facts about games that are often overlooked because of their isolation from traditional UI design. First, games operate on the same principles as other interactive systems, they share the same design criteria of effectiveness, efficiency and satisfaction, and they have had to solve many of the same interface problems that conventional applications face. Second, games are no longer played only by teenaged boys – 40% of frequent PC game players are older than 36, and 38% are female – and the 'gamer demographic' is becoming indistinguishable from that of the 'ordinary' users who use conventional applications.

Although we believe that the innovations we have introduced will help to improve the usability of conventional applications, there are issues of applicability and risk, particularly in that some of the ideas radically change the outlook of the application. In the next sections, we review some of the issues for each of the four main areas discussed earlier.

Effortless community. Porting the concept of natural community to conventional applications poses three main challenges: distraction, privacy, and security. Synchronous collaboration has the potential to be distracting, and can interfere with users' individual work. There are also substantial privacy and security issues involved with adding this type of support, particularly when the user's workspace contains sensitive or proprietary content. Although these issues must definitely be addressed, they go with the territory. The advantages of collaboration and community must be weighed against the compromises, but there are many situations where the benefits will outweigh the risks.

Learning by watching. Two main challenges in porting observational learning to applications are privacy and visibility. People will be less comfortable with letting others watch them work than they will when playing games. A carefully designed protocol for requesting and granting permission to be observed is a prerequisite for this

type of technique to be effective and acceptable to users. The other challenge is being able to show the observer what is happening in a rich way that does not hinder the expert. Nevertheless, the technique has great potential because it is very beneficial to the observer while requiring little or no work from the expert, something that many groupware systems have difficulty with [6].

Deep customizability. Customizability in games is typically used to increase performance on common tasks and to accommodate user preferences. This goal is quite general, and for that reason, the customizability concepts we have presented in this paper can be reasonably transferred to a range of applications. While better performance is desirable, there are potential tradeoffs in memorability and consistency. Additionally, it is not entirely clear whether users of more conventional applications would embrace the level of customizability that is available in games. For example, it is not clear how scriptable interfaces and portable customizations would be used or accepted by users of conventional applications, and whether these users would be as motivated as game users to modify the UI.

Fluid system-human interaction. There are a number of risks associated with implementing the approaches games take to fluid system human interaction in other applications. First, it is unclear whether using sound rather than written text to convey information would be too distracting in office settings. Also, the use of sound and fading messages can cause important information to be missed, since these messages are not persistent. Finally, it is not clear how users will respond to context aware view behaviours and whether they will be willing to relinquish total control of their view to the system.

There seem to be many instances where these techniques could increase performance through a more fluid workflow. The challenge now is to determine where and when these techniques are appropriate.

CONCLUSION

Our exploration to find design innovations in games has identified radical and novel interaction concepts and has produced a wealth of ideas for future work. Games have already shown that these approaches are beneficial to the user through success in real-world use. The potential for applications to benefit from adopting these novel contributions is realistic because games and applications share many commonalities. Further research into how these innovations can be generalized, and a continued interest in the progress of the game domain, will hopefully lead to usability benefits for users of all applications.

REFERENCES

1. Abowd, G.D., Beale, R. Users, systems and interfaces: a unifying framework for interaction. In HCI'91: People and Computers VI, pages 73-87. 1991.
2. Cox, D.A, Chugh, J.S., Gutwin, C., Greenberg, S. The usability of transparent overview layers, Proc. CHI'98, pp. 301-302.
3. Donath, J.S. Visual Who: animating the affinities and activities of an electronic community, Proc. ACM Multimedia 1995, pp. 99-107.
4. GameSpot.http://gamespot.com/gamespot/misc/userreview/explained.html
5. Gaver, William. W. "The SonicFinder: An Interface that Uses Auditory Icons." Hum.-Comp. Inter. 4(1) (1989).
6. Grudin, J. Groupware and social dynamics: eight challenges for developers. Communications of the ACM, 37(1), 1994, ACM Press, p.92-105.
7. Gutwin, C., Greenberg, S. Design for individuals, design for groups: tradeoffs between power and workspace awareness. Proc. CSCW'98, pp. 207-216.
8. Gutwin, C., Greenberg, S., Roseman, M. Workspace awareness support with radar views, Proc. CHI'96, pp. 210-211.
9. Harrison, B.L., Vicente, K.J. An experimental evaluation of transparent menu usage, Proc. CHI'96, pp. 391-398.
10. Hindmarsh, J., Fraser, M., Heath, C., Benford, S., Greenhalgh, C. Fragmented interaction: establishing mutual orientation in virtual environments, Proc. CSCW'98, pp. 217-226.
11. Hopkins, D. The Design and Implementation of Pie Menus. Dr. Dobb's Journal, December, 1991, pp. 16-26.
12. Interactive Digital Software Association, Essential facts about the computer and video game industry, available at http://www.idsa.com/IDSABooklet.pdf
13. Mackay, W.E. Triggers and barriers to customizing software, Proc. CHI'91, pp. 153-160.
14. Malone, Thomas W. "Heuristics for designing enjoyable user interfaces." Proc. of the first major conference on Human factors in computers systems March 1982.
15. Norman, D.A. The Psychology of Everyday Things. Basic Books, New York, 1988.
16. Tan, D.S., Robertson, G.G, & Czerwinski, M. Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting. Proc. CHI 2001, 498--505.
17. Whittaker, S. Talking to strangers: an evaluation of the factors affecting electronic collaboration, Proc.CSCW'96, pp. 409-418.
18. Wolf, C.C. A comparative study of gestural, keyboard, and mouse interfaces. Behaviour and Information Technology, 1992, v.1(1), pp. 13-23.
19. Woods, D.D. The price of flexibility, Proc. Intelligent user interfaces, 1993, pp.19-25.