Information Visualisation Utilising 3D Computer Game Engines Case Study: A source code comprehension tool

Blazej Kot	Burkhard Wuensche	John Grundy	John Hosking
bkot002@ec.auckland.ac.nz	burkhard@cs.auckland.ac.nz	john-g@cs.auckland.ac.nz	john@cs.auckland.ac.nz

Department of Computer Science

The University of Auckland

ABSTRACT

Information visualisation applications have been facing ever-increasing demands as the amount of available information has increased exponentially. With this, the number and complexity of visualisation tools for analysing and exploring data has also increased dramatically, making development and evolution of these systems difficult. We describe an investigation into reusing technology developed for computer games to create collaborative information visualisation tools. A framework for using game engines for information visualisation is presented together with an analysis of how the capabilities and constraints of a game engine influence the mapping of data into graphical representations and the interaction with it. Based on this research a source code comprehension tool was implemented using the Quake 3 computer game engine. It was found that game engines can be a good basis for an information visualisation tool, provided that the visualisations and interactions required meet certain criteria, mainly that the visualisation can be represented in terms of a limited number of discrete, interactive, and physical entities placed in a static 3-dimensional world of limited size.

Categories and Subject Descriptors

H.1.2 [**User/Machine System**]: Human factors; H.5.2 [**User Interfaces**] Graphical User Interfaces (GUI), Interaction styles; H.5.3 [**Group and Organization Interfaces**] Collaborative Computing; I.3.6 [**Methodology and Techniques**] Interaction Techniques; I.3.8 [**Computer Graphics**] Applications; K.8.0 [**General**] Games.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CHINZ '05, July 6-8, 2005 Auckland, NZ

Copyright 2005 ACM 1-59593-036-1/04/10"

General Terms

Algorithms, Human Factors.

Keywords

Information visualisation, human-computer interaction, game engines, collaborative visualisation, software visualisation.

1. INTRODUCTION

Modern computer games make use of technologies from almost all areas of computer science: graphics, artificial intelligence, network programming, operating systems, languages and algorithms. A modern computer game engine, such as Doom 3 [13] or Unreal Tournament 2004 [15] contains efficient, very well-tested implementations of a wide range of powerful information visualisation and interaction techniques. These are generally focused on rendering realistic 3D "worlds" and supporting navigation within and interaction with elements in the visualisation. Given the power, flexibility and maturity of these game engines, we wanted to investigate possible ways of reusing these implementations for other information visualisation tasks, thus potentially saving large amounts of development time.

Computer game implementations have been successfully applied by other researchers to tasks they were not originally designed for, such as architectural design critique [1], military simulations [2], landscape planning [3], and as an interface for Unix process management [4]. In these applications game engines were utilised to render very domain-specific information spaces which were then navigated and interacted with by users.

Our research focuses on utilising computer game implementations for more general information visualisation tasks. This seems to be a relatively unexplored area, as we could only find a few examples of work related to this topic [4,5]. We present a framework for information visualisation using game engines and we apply it in practice by developing a source code comprehension tool based on the Quake 3 engine. This example was chosen because code comprehension requires the user to explore a structure, and to remember the locations of various items in it. Hence this information can be mapped into a 3D environment, where the spatial memory of the user can be engaged to remember the layout of the code structure.

We provide a motivation for this research and survey game engine technologies and related research. We then introduce an architectural approach to information visualisation using games engines. We illustrate the application of this architecture to the code comprehension information visualisation domain, describe a proof-of-concept prototype we developed to illustrate its feasibility, and describe an evaluation of this prototype. Further details of the research presented in this paper are available in [11].

2. MOTIVATION AND BACKGROUND

2.1 Source Code Comprehension Tools

Source code comprehension involves taking a complex data set (program source code), translating this into a data model with relationships between source code items represented (which might include type dependency, operation invocation, data flow, thread synchronisation, resource utilisation and so on), and visualising this data model. Users (typically designers and programmers) often want to visualise the data model in various ways, navigate the model from high-level to low-level relationships and vice-versa, and perform tasks on selected data model items e.g. viewing a file [29]. Most existing source code comprehension tools use textual or 2D diagrammatic renderings, with associated navigation and interaction paradigms. A few examples, like Shrimp [30] and Bloom [31], use 3D, zoomable renderings. However, these tools are built with bespoke techniques and require ad-hoc design and implementation techniques to realise their 3D user interfaces. Most existing tools provide very limited support for multi-user source code comprehension tasks.

We wanted to develop a source code comprehension tool that used 3D, virtual environment metaphors to both present source code relationships and support end user browsing and interaction with information items. We also wanted to support multiple users working together to comprehend code, including code walk-throughs, code annotation and communication. To avoid building our own support for these features from scratch we wanted to leverage game engine technologies. Thus our tool needed to use a visualisation metaphor that maps simply and naturally into a static 3D world occupied by dynamic, interactive entities, as found in most current game engines. Key functional requirements of the tool included: rendering source code files as 3D entities in a suitable 3D world; relationship rendering and hyperlinks between dependencies; support for multiple user interaction, including identifying places of current interest of other users, guided tours of code, and on-line communication in the virtual world.

2.2 Game Genres

Out of the computer game genres which use graphics (as opposed to text-based games) the main ones are First Person Shooter (FPS), Real Time Strategy (RTS) and Role Playing Game (RPG). Computer games can be classified further into single player (wherein other players are simulated using artificial intelligence), multi-player (where several players can interact in the same virtual world via a computer network), or both.

In a FPS game, the player travels around in a three dimensional world, shooting enemies. In a RTS game, the player views a two dimensional map with many units (for example, of an army) on it. Players control their own units and use them to attack and defeat their opponents. A RPG is similar, except the player controls only one unit, their “character”, via which they explore a 2D (e.g. Age of Empires II [6]) or 3D (e.g. World of Warcraft [7]) world.

2.3 Game Architecture

Most modern computer games can be split into three parts: the game engine, the game logic and the game art. The game engine is the main executable file which runs on the computer. It provides an environment within which the game logic runs, as well as basic mathematics, graphics, audio, user input and network functions. The game logic may take the form of scripts, bytecode for a virtual machine, or a library (a DLL for example). The game logic's task is to control the game play, and to use the engine to display the game art as appropriate. The game art consists of things such as pictures (textures in game parlance), maps (layouts of virtual worlds), models (3D representations of things inhabiting the world, for example players, weapons or flowerpots) and sounds.

2.4 Available Game Engines

Game engines can be divided into two categories: open source and closed source.

Open-source game engines are either ones written by amateurs, or older commercial engines which the developer decided to open-source. In the former category, some of the more popular engines are: OGRE [8], Crystal Space [9], Irrlicht [10], and The Nebula Device 2 [12]. In addition to these, there are the Doom, Doom 2, Quake and Quake 2 engines which have been open sourced by id Software [13]. These use the OpenGL library for rendering, and since they are from a commercial game they include support for all the gaming features such as physics, audio, network, 2D and GUI. None of the four amateur engines mentioned above have as much functionality as the id Software engines built-in, however they may be combined with external libraries to provide the missing functions. The id Software engines suffer from being older, providing poorer rendering quality than the newer open source engines.

There are currently three main closed-source game engine families in the FPS genre, each from a different developer: Doom 3 and Quake 3 engines from id Software, Half Life and Half Life 2 engines from Valve Software [14] and Unreal Tournament (UT) and Unreal Tournament 2004 (UT2004) engines by Epic Games [15]. Doom 3, Half Life 2 and UT2004 represent the latest generation from each developer, and are the best game engines available.

All of these closed-source engines are fully-featured, and there exist one or more complete games based on each of these engines. This is in contrast to most of the amateur engines mentioned above, which provide more basic functionality. The amateur engines often need to be combined with other libraries and toolkits to create a playable game, while the six closed-source engines listed here have all of the required functions built-in. Out of these six, Quake 3 deserves special mention as id Software plans to open source it in the near future [16]. This would mean that, unlike the other engines here, extensive modifications to the game engine would be possible. (Mods for the other ones are restricted to altering the game logic and art.)

3. RELATED WORK

3.1 Using Game Engines for Visualisation

PSDoom [4] is a utility for process visualisation and management, implemented as a modification of the Doom computer game. It provides the functionality of the Unix `ps` command via a 3D user interface. Running processes are represented as monsters (enemies), which can be shot and killed, thereby terminating the associated process. Monsters can fight back, and more important processes are represented by bigger monsters (which are more difficult to kill), thereby reducing the chance that they will be terminated. Interestingly, when many processes are running, and the 3D space becomes crowded with monsters, the monsters start

attacking each other (a normal Doom behaviour). This provides a natural control mechanism for processes in a heavily loaded system - less important monsters will be killed first, since the important monsters are represented by stronger monsters.

Heckenberg et al. [5] implement a visualisation of a simplified financial market ("The Minority Game") using a modification of the Unreal Tournament 2003 computer game. The Minority Game consists of two teams of agents. These agents are represented as 3D players in the game. The Minority Game centres on the agents making decisions so as to end up on the winning team (which is defined as the team with the least agents). Future work is suggested to make use of other game features, such as players being able to throw stocks and money at each other, to perform trading.

Computer games can also be used to visualise less abstract concepts, such as 3D battlefields in military simulations [2], landscape design and planning [3], and architectural designs [1].

3.2 Software Visualisation

A substantial amount of research has been done in the area of software visualisation using 3D graphics. The tool sv3D [22] represents one source file as a group of cuboids, one for each line of code. The colour of the cuboid may represents various attributes of the corresponding line, such as the control structure type to which this line belongs, while the height can represent the nesting level of the line. The sv3D implementation allows the user to arbitrarily map other attributes of the code lines to various graphical attributes, such as transparency.

Another interesting 3D visualisation is ArchView [23]. Here, individual source code modules are represented by "LEGO blocks" floating in a 3D space. Import relationships between modules are shown by arrows, while the type of module (a user-specified categorisation) is represented by the colour and shape of the bricks.

Panas et al. [24] use a 3D city metaphor to represent a software project. Java classes are represented by individual buildings, whose size represents the number of source code lines in that class. The spacing of buildings shows the amount of coupling between the classes, and the type of building indicates the quality of the code - old and collapsed buildings represent code which needs to be refactored. Cars travelling within this city show the dynamic execution path of the visualised program. Since the cars leave traces, places of heavy traffic can be identified, and these correspond to heavy communication between classes. Various aspects related to software project management are then superimposed on top of this view, such as surrounding often executed classes by flames and colouring parts of the code which are not used brown.

4. INFORMATION VISUALISATION USING A GAME ENGINE

4.1 The Visualisation Pipeline

The visualisation process can be represented by a pipeline which performs a *data encoding* and a *data decoding* step as shown in figure 1. The first stage of the data encoding step is the *data transformation* stage that converts information into a form more suitable for visualisation. This can involve the creation of new quantities and subsets, data type changes, and modelling operations (e.g. model a directory structure as a tree). The subsequent *visualisation mapping* converts the transformed data into graphical representations which the *rendering* stage then displays on a screen or by printing.

For information visualisation applications some authors [27,28] prefer to subdivide the mapping stage further into *visual transformation* (or *data modelling*) and *visual mapping*. However, in many applications these two stages are combined: the available models are fixed and the parameters of a model (shape, size, colour, texture) represent the encoded information. The data decoding step describes how visual information is perceived and processed and consists of *visual perception* and *cognition*.

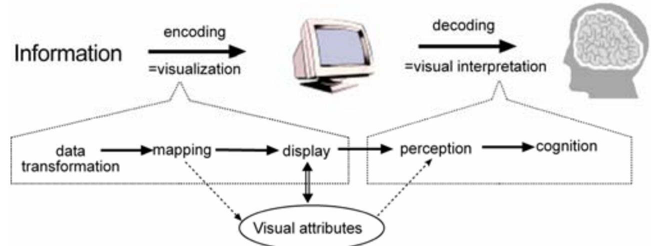

Figure 1: The Visualisation Pipeline.

The encoding and decoding are connected via *visual attributes* such as shape, position, and colour, and *textual attributes* such as text and symbols which themselves are represented by simple visual attributes. A visualisation is effective if the decoding can be performed efficiently and correctly. "Correctly" means that perceived data quantities and relationships between data reflect the actual data. "Efficiently" means that a maximum amount of information is perceived in a minimal time.

4.2 Integrating a Game Engine into an Information Visualisation Framework

There are two main ways in which a FPS game engine can be used for information visualisation. One way is to modify an existing game which is implemented on top of the engine, and only add the features necessary for the visualisation, leaving the basic style of interaction with the 3D world intact. The other way is to write totally new code for the game logic, and only make use of the graphics, audio and networking functionality provided by the engine itself. This approach is more flexible with regards to what visualisations can be created, however it requires a lot more work on the part of the developer. In fact, this approach is similar to using a visualisation toolkit or engine, such as OpenSG[25]. In this paper, only the former approach is considered, as this is the option that allows maximal reuse of the computer game implementation.

4.3 Information Mapping

The main challenge met when using a game engine for information visualisation is that the set of available visual attributes, textual attributes and interaction techniques is limited as explained below. Hence it is imperative to take these limitations into account when transforming and mapping the raw data into graphical representations.

In an FPS game, there are two primary types of elements: a static, or almost static, map (3D layout of rooms) and dynamic, interactive entities occupying positions in this map.

There are many different ways to represent parts of an information visualisation by game elements. The particular

mapping chosen depends on the particular visualisation. For example, in a visualisation of a file hierarchy, the layout of directories could be represented by the layout of the rooms (that is, the map), while files are entities occupying positions within these rooms.

Limitations imposed by game engines must be taken into consideration when designing this mapping. One example is that in most current FPS games (specifically, Quake 3), the map can not be altered during a game session. This could be partly worked-around, as players can be moved between maps relatively easily, so that one map could be altered while the players are in another map, creating the illusion of a dynamic world. This has the disadvantage that the game will pause while switching maps. Additionally, Quake 3 maps are limited in size. A solution is to split a large map into several smaller maps, but with the same problem of the game pausing between map changes.

Another limitation of FPS games is that they are designed for a relatively low number of entities; Quake 3 only allows a maximum of 1024 entities in a map. With access to the game engine source code, this limitation may be removed, but this may introduce a performance hit. Thus, in some cases it may make more sense to represent parts of the visualisation as dynamically generated textures (e.g. a diagram of a graph structure) rather than as separate entities. Again, this could be worked around by using multiple maps, or by spending some time extending the engine.

Yet another peculiarity of game engines is that they are designed to only support one style of interaction, that defined by the game logic. For example, in Quake 3 each entity usually has a fixed appearance, and a fixed behaviour throughout a game session. (It is actually possible to alter these programmatically during a game session in the game logic, if desired.) The problem is that the engine does not provide any "multiple view types" support. So, if the visualisation to be implemented relies on multiple view types (e.g. seeing files first as parts of a pie chart of disk usage, and then as entities inhibiting 3D rooms), one must be prepared to code a framework on top of the engine which will keep track of what view is being currently used, and tell the game logic which representations and behaviours to use for which entity.

5. A SOFTWARE COMPREHENSION TOOL BASED ON A GAME ENGINE

5.1 General Requirements

The initial choice we faced was how to map important features of code visualisation to a 3D game engine metaphor. The metaphor we chose is to represent source code files as entities that can be interacted with (viewed, moved, arranged) in an otherwise static 3D world. Although we have chosen source files as the principal entity, we could easily have chosen class definitionsor any other primary modularisation mechanism to similar effect. File entities can be moved around at will, to arrange them in logical groupings. They can be viewed by walking up to them. The source files are cross-referenced using hyperlinks, so that clicking on a symbol in the source code takes the user to a definition of that symbol. Also, back and forward buttons exist so that users can walk through their histories. The metaphor used by the tool is essentially just a 3D version of a web browser, using cross-referenced source files as the web pages. The advantages of using a game engine instead of a normal browser are mainly: multi-user support either via a LAN or the internet, so one user can guide another user via the code; and utilisation of the user's spatial memory, since closely-related files can be placed together in the 3D world - hopefully making it easier for the user to comprehend the structure of the source code.

It is important to note that care must be taken when using 3D interfaces for information visualisation applications. Results of Cockburn et al. [21] suggest that users often find a 3D user interface confusing and cluttered. However, the authors also mention that adding semantic labels to elements in a 3D space may improve user performance to above that of a standard 2D interface. The tool implemented in the present project places the elements in virtual rooms and hallways in a 3D world, unlike the experiment by Cockburn et al. where the items were all floating in space. The implementation hence provides a type of semantic label and utilises the users' spatial memory abilities [19,20].

Throughout the design of the tool, the main usage scenario considered was that of a new employee, or group of employees, arriving to work at a company, and being given a guided tour of the company's source code base by an existing employee. The main functional requirements thus identified included:

- A web browser like interface, but with webpages (source code files) represented as movable 3D entities in a 3D world. Syntax highlighting and hyperlinks are to be displayed when a source code file is shown.
- Ability for users to identify, by their appearance (model), other users in the world.
- Ability for a user to give a guided tour - so that new employees can automatically follow experienced employees.
- Ability for users to point out specific parts of the code to each other during a guided tour - by circling parts of code on their screen, and by having the same drawing appear on the other player's screens.
- Communication between users (via a chat facility)

5.2 Requirements for the Game Engine and Engine Choice

The game engine used needs to be multi-user capable, stable, and well tested. Since as discussed in Section 4 the tool will be implemented by modifying an existing game running on the chosen game engine, there must be a well-tested, open-source, implementation of a game for the chosen game engine. This is required since this project aims to reuse as much of the 3D First Person Shooter style of interaction as possible. Because of this, it makes sense to reuse an existing FPS implementation, rather than spending time writing one for a game engine which has no such existing implementation.

The game engine which seemed to best meet these requirements is the Quake 3 engine, with the corresponding game implementation, Quake 3 Arena [17,18],. The Quake 3 engine source code is at the moment not available to the public. The Quake 3 Arena source code is available, under a limited licence (which does appear to permit modifying the source code and distributing the modified game virtual machine bytecode).

Several open source game engines, such as OGRE, Crystal Space and Irrlicht (see section 2.3), were investigated, however most of them lacked crucial features (such as networking support), or had no well-tested game implemented using them.

Quake 3 uses the standard FPS game control system: mouse and keyboard. Moving the mouse around changes the direction the player looks in. The mouse buttons are typically used for walking forwards and for shooting. Various keys on the keyboard are used for crouching, jumping, moving backwards, strafing and switching weapons. The keys can be remapped to different in-game functions via a process known as *key binding*.

Quake 3 is by design a network-oriented (LAN or internet) game, using the client-server model of communications. Each computer running Quake 3 runs an instance of `quake3.exe`, the game engine. This executable is capable of running bytecode for three virtual machines: `game`, `cgame` and `UI`. These are referred to as *QVM*s, for Quake Virtual Machine. The `game` qvm is the server part of the game. It is responsible for maintaining the state of the game world, such as positions of all entities, and sending messages to the clients. It also has the final say on issues such as whether a certain bullet hit a certain player or not. `Game` does not do any rendering; it only communicates with clients. `cgame` is the client QVM - there is one running on each computer connected to a particular game. The client is responsible for rendering the map and entities, according to data sent by the server. Finally, the `UI` qvm is responsible for displaying the in-game menus. Within the QVM environment there are several available system calls or *traps*. These are functions that can be called within the code of the QVM, to pass control into the main `quake3.exe` executable. This is how tasks such as drawing on the screen, file and network access are carried out.
Internally, the main message passing mechanism (or rather, the most easily accessible and modifiable one) is based on passing variable-length, null-terminated strings. These can be sent from the server to the client or vice-versa.

Currently, only the Quake 3 game logic code is open to the public. This consists of a total of approximately 100 000 lines of ANSI C code running on the three Quake 3 virtual machines.

5.3 The User Interface and Interaction Metaphor - Single User

The tool is based on displaying source code files as individual entities within a 3D world. These entities can be moved around, much like icons can be moved around on a 2D desktop. The entities are drawn as floating "T" shapes, for no reason other than that this seemed to be the most appropriate of the existing entity models in Quake 3. Another model could be easily substituted in the future. The size of the drawn entity indicates the size of the corresponding source file. All file entities have their filename displayed above them. In order to improve the readability this is done in a way such that the text always faces the player, and is of constant size, irrespective of the distance between the player and the file. Additionally, header files (detected as those files whose name ends in ".h") have their filenames displayed in red, and their "T" shapes surrounded by a spheroid, while other files have their filenames in blue, and have no surrounding spheroid, so that they may be easily distinguished as illustrated in figure 2.

Files can be picked up by shooting at them. When this occurs, the file disappears from the map, in all players' views, but a "T" icon appears on the side of the screen of the player who is now holding the file. The player can then walk anywhere on the map, and press the item use key to drop the file at the new location. The file than appears at this new place in the map. In the current implementation, there is no way for players to distinguish if a certain player is carrying a file. This may cause some confusion due to "missing" files, so this feature should be added in a later version.

Figure 2: A small header file (left) and a large .c file.

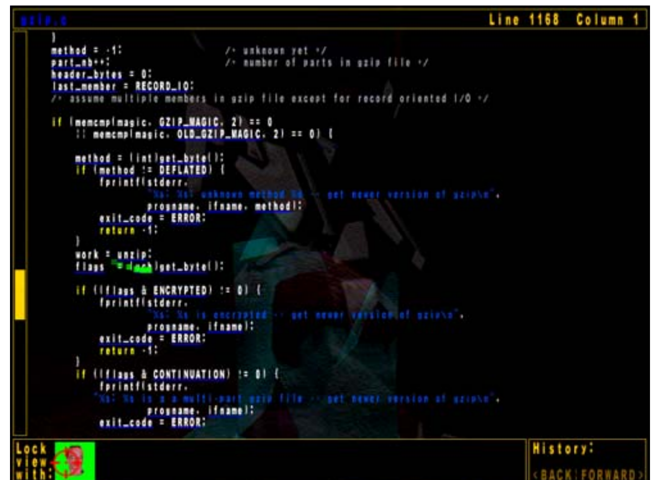

Figure 3: Viewing a source code file.

To view the contents of a file, players just walk into a file. When they are close enough, the file is displayed on the screen, as shown in figure 3, in a scrollable box. The file may be scrolled using the arrow and page up and down keys. The current position in the file is indicated by a scrollbar on the left and as line and column numbers at the top of the screen.

The displayed text is shown using syntax highlighting, for easier comprehension. Function invocations and uses of variables are displayed as hyperlinks - by being underlined in blue. The hyperlinks are generated by using the doxygen [26]

documentation tool, in conjunction with a purpose-written parsing tool. Clicking on a hyperlink will take the user to where the symbol in question is defined in the source code. If the definition is within the same file, the file is scrolled so that the definition is at the top of the screen. If the definition is in another file, the file view is closed, so that the user sees the 3D world. The user's view is than slowly panned around so that the file in which the symbol is defined is centred on the screen. The player is then slid towards the file, and upon reaching it the file is displayed, with the definition of the symbol at the top of the screen. In the present implementation, if the files are in different rooms separated by a wall, the player will be slid through the wall, which may be disorientating to the player. In future, part of the AI route-finding algorithm in Quake 3 could be reused to have the player follow a path to the destination file without crossing any walls.

Each time a player clicks on a hyperlink, the currently viewed file, and their position therein, is added to their history. The history operates exactly like a web browser's history, via a back and forward button.

5.4 Multi-User Features

Most of the multi-user features present in Quake 3 remain unaltered. These include the ability of players to see other players in the 3D world, to chat with them, and to view the names of all players in the current game by pressing the F1 key (the scoreboard key). In Quake 3, each player can choose a 3D model that will be their avatar in the 3D world. Each player also can set their name - this is the name that will be displayed in the scoreboard and in the chat dialog.

The standard Quake 3 interaction provides a simple way of seeing what file another player is looking at - one can walk around the map, looking for that player's model, and see what file they are standing at. Unfortunately, the standard Quake 3 multi-user system does not provide a simple way of seeing which part of the file another user is looking at, and also there is no simple way of pointing out certain lines of code to another user.

Therefore, two new multi-user features were added to Quake 3 for this tool: a way of "locking" one's view with another player's, so that your screen shows exactly what the other player is seeing; and a way of "drawing" on the file display, so that one can point out parts of the source code to other users.

The lock view feature works as follows: when viewing a file (ie. when standing very near to it), portraits of all other players also viewing the same file are displayed at the bottom of the screen. Clicking on a portrait locks your view with that player's. This is indicated by the background of that player's portrait flashing. When your view is locked with another players', you cannot scroll, but your view shows exactly what the other player sees - if they scroll, your view also scrolls to the same position.

One can point out parts of source code to other users currently viewing a file. This is done by dragging the mouse cursor with the right mouse button held down. This leaves behind a long trail of squares, which fade out over time. This can be used for example for circling or underlining certain items. In the present implementation, if one player viewing a file draws on it, the same trail of squares appears in the view of all other players also viewing the source file, irrespective of whether their views are locked or not. Whether the trail drawing should be restricted to only the players with locked views is something to be investigated in the future. Each player has a unique colour, which is used as the background of their portrait, as well as for the colour of the trail of squares. This is so users can easily identify who is drawing. See Figure 4.

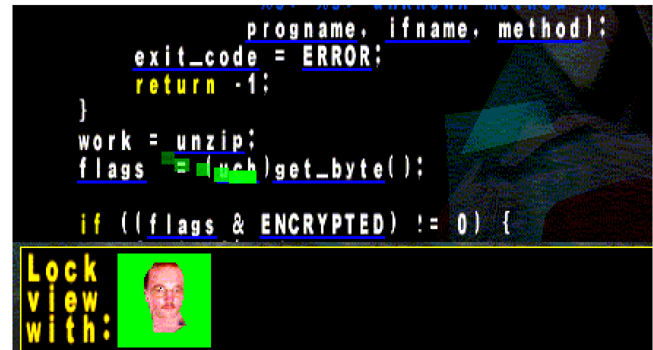

Figure 4: Detail of the source file view. The player with the green background is pointing out the "unzip" link using a trail.

6. EVALUATION

6.1 Evaluation of the Tool

The implemented tool met all the major functional requirements listed in the design section: source code files are represented as physical, movable 3D entities in a 3D world. They can be navigated via a web browser like interface which displays the files with syntax highlighting and hyperlinks. Users can see each other, and can customise their appearance to be unique using the normal Quake 3 model choosing system. A player can lock their view with another player, so that giving guided tours is possible. Users can "draw" on the source files, thereby indicating important parts of the code to other players also viewing that file (the "drawing" fades over time to reduce clutter). The chat functionality is not completely implemented yet, as it does not work correctly for users who are viewing contents of files, however it does work well in the normal 3D view.

There was no real usability trial done. However, during debugging, the help of several people was enlisted, and the following observations were made:

- Since everyone was in the same physical room, no-one bothered using the chat functionality, and just communicated using voice.
- Sometimes in a place where there were lots of files, the file name labels would overlap, being difficult to read. One person came up with an interesting use of the Quake 3 'zoom' function to overcome this. By holding down the 'Ctrl' key, the player's view is magnified, showing less files, and separating the filenames labels more, making them easier to read.
- An unplanned feature was discovered: a player who is currently carrying a file can run up to another player, and drop the file onto him. The other player's view will then show the contents of that dropped file. It is interesting to note that this feature arose by itself due to basing the tool on a 3D physical world metaphor. In the future, this sort of interaction could be used, for example, for a 3D interface to a source code version control system: a user walks up to a

"vault", where the checked-out files are dispensed, picks one up, carries it to their virtual workplace to work on it, and then carries it back and "drops" it into a chute leading back to the vault, to check-in the changes.

- In an early version of the tool, the lock view feature was only implemented within one source file view - as soon as the tour guide clicked on a link, the other participants' views became unlocked, and they did not automatically follow the tour guide to the destination of the clicked link. The participants found this annoying, so the lock view feature was changed in the current version, to allow groups to follow the leader between files automatically. Unfortunately, this seems to have had the effect of turning the players who are being guided around into "zombies" - all that they do is look at the screen, and have no control of what they are looking at. This may lead them to losing attention quickly. Perhaps an intermediate solution could be implemented, where if the leader clicks on a link, that link is highlighted on the other player's screens, but then they have to click on it. Whether this is a good solution needs further investigation.

6.2 Evaluation of Quake 3 Game Engine Suitability

We found that Quake 3 was a good choice for the underlying implementation platform. Some minor problems were encountered with using this engine as listed below.

Learning to modify the game engine took time, and was mostly done through trial and error. There is no documentation of the code provided by idSoftware. There are basic third party resources [17,18] that serve as a good introduction to the basics. However, it was often necessary to walk through the existing code by hand and figure out how it worked and how best to modify it.

The game code which runs on the VMs is written in C. This has the usual side-effect of having the implementation of a particular entity split across many different files. This is not necessarily a bad thing in itself, but sometimes this leads to subtle bugs where some other part of the code in another file unexpectedly alters the state of an entity.

The lack of dynamic memory allocation in the VMs is a limitation, but can be easily worked around. Either a large static array can be used and memory allocated out of that by hand (as was done in the cgame.qvm modification), or once the game engine source code is opened, this functionality may be added. However, such an addition would need to be carefully planned, since presumably there was a good reason for not including this functionality in the first place. Another way to work around this issue, which is not optimal, is to compile the game code as a win32 DLL. This can be loaded by Quake 3, and can make use of all native win32 functions such as dynamic memory allocation. However, this solution is non-portable, and removes the security barrier which is provided by executing the code in a QVM.

There seems to be a bug in the game engine itself, whose source code is not currently available to the public. This bug manifests itself when a large number (> 8000) of triangles are drawn from within the UI QVM. This is an important example of how using the engine for what it was not designed for (displaying lots of text in the UI) may test the engine in ways that a normal game wouldn't, revealing hidden bugs. Since the Quake 3 game engine should be open-sourced soon, this bug should be able to be fixed.

Another result of the game engine code being, for the moment, closed, is that the network protocol is fixed. New character string messages can be added easily, but these need to be assembled at one end, and parsed at the other. In addition, these messages are not associated with any particular entity (other than, in the case of client-originating messages, which player sent them). This was a problem in the implementation of the tool since the new `itemid` field of `sourcefile` entities had to be communicated from the server to the client. Fortunately, a field in the existing packet structure was found which was rarely used, and so was reused to transmit the `itemid`. While this worked for this particular tool, the available space is very limited, and may not be sufficient for other visualisation tools. A workaround based on sending the extra information via text messages and storing them in a look up table at the destination could be implemented, or, better, the network protocol could be made more flexible once access to the game engine source code is available.

The last significant problem encountered was the limited capability of the UI system built into the game, specifically that it did not have a text scroll box element. However, all functions needed for implementing such an element were easily accessible, so there were no problems with coding this by hand (other than the 8000-triangle limit mentioned above).

7. CONCLUSION

It is possible to use a game engine as the basis for an information visualisation tool, and thereby save a lot of implementation time by reusing the functionality already implemented in the game engine. The biggest benefit is obtained when as much of the game engine functionality is reused as possible, i.e. in 3D, interactive, multi-user visualisations based on a metaphor involving physical entities.

In order to simplify the implementation process, it is important to choose a visualisation which uses a metaphor of there being several distinct dynamic entities in a static 3D world that the users can interact with. There should typically be much less than 500 of these entities, but this limit depends on the game engine chosen, the amount of modifications needed, and the performance and hardware requirements of the result. There typically is also a limitation on the maximum allowed map (3D world) size, although this can be usually worked around by stitching together several smaller maps.

Out of the two ways of reusing a game engine, those being either writing a new set of source code which uses the engine, or modifying an existing game to customise it for producing the desired visualisation, the second is much easier and was therefore used in this project. There is still considerable work involved in customising an existing game, especially as in this case, where there was very little documentation of the code available.

The tool which was implemented in this project seems to work well as a source comprehension tool, this observation being based on an informal trial. It closely resembles browsing a cross-referenced source code base via a web browser, with the addition of multi-user capability and the ability to use one's spatial memory to remember the structure of the code.

8. FUTURE WORK

The implemented tool may make a good basis for future software visualisation projects, wherein various graphical representations of information could be superimposed on top of the existing

visualisation. If users have already stored the basic structure of the source code files in their spatial memory, they could easily identify what parts of the source code the superimposed information relates to. It may also be worth investigating the use of a tool similar to this for browsing other repositories of hyperlinked documents - for example, using hyperlinks to show citations between scientific papers.

The tool itself could be extended in several ways, such as adding support for "notes", which could be used to annotate the 3D spaces with information about what function the source code files placed there perform. Also, the map generation could be automated by analysing the source code. Further improvements could include VOIP support for voice chat capability, and adding a 2D overview map so that the location of files and other players could be quickly ascertained. Other future research could involve seeing how well game engines of other genres (RPG, RTS) are suited for information visualisation.

9. ACKNOWLEDGMENTS

We gratefully acknowledge that this research was supported by a grant from the Faculty of Science of the U of Auckland.

10. REFERENCES

[1] Moloney, J., Amor, R., Furness, J., and Moores, B. Design Critique Inside a Multi-Player Game Engine, *Proceedings of the CIB W78 Conference on IT in Construction*, Waiheke Island, New Zealand, 23-25 April, 2003, pp. 255-262.

[2] Manojlovich, J., Prasithsangaree, P., Hughes, S., Chen, J. and Lewis, M. UTSAF: A Multi-Agent-Based Framework for Supporting Military-Based Distributed Interactive Simulations in 3D Virtual Environments, *Proceedings of the Winter Simulation Conference*, New Orleans, LA, 7-10 December, 2003, pp. 960-968.

[3] Herwig, A., Paar, P. Game Engines: Tools for Landscape Visualization and Planning?, *Trends in GIS and Virtualization in Environmental Planning and Design*, Wichmann Verlag, Heidelberg, 2002, pp. 161-172.

[4] Chao, D., Doom as an Interface for Process Management, Proceedings of SIGCHI'01, Seattle, WA, 31 March-1 April 2001, pp. 152-157.

[5] Heckenberg, S.G., Herbert, R.D. and Webber, R. (2004). Visualisation of the Minority Game Using a Mod. In Proc. *Australasian Symposium on Information Visualisation, (invis.au'04)*, Christchurch, New Zealand. Conferences in Research and Practice in Information Technology, 35. Churcher, N. and Churcher, C., Eds., ACS, pp. 157-163.

[6] Age of Empires II, www.microsoft.com/games/age2.

[7] World of Warcraft, www.worldofwarcraft.com.

[8] OGRE Object-oriented Graphics Rendering Engine, www.ogre3d.org.

[9] Crystal Space 3D, crystal.sourceforge.net.

[10] Irrlicht Engine - A free open source 3d engine, irrlicht.sourceforge.net.

[11] Blazej Kot, " Information Visualisation Utilising 3D Computer Game Engines ", FoS Scholarship Report, University of Auckland, February 2005, www.cs.auckland.ac.nz/~burkhard/Reports/SS2004_BlazejKot.doc

[12] The Nebula Device 2, nebuladevice.cubik.org.

[13] id Software, www.idsoftware.com.

[14] Valve Corporation, www.valvesoftware.com.

[15] Epic Games, www.epicgames.com.

[16] John Carmack's Blog, www.armadilloaerospace.com/n.x/johnc/Recent%20Updates.

[17] Quake III: Arena, baseq3 mod commentary, www.icculus.org/~phaethon/q3mc/q3mc.html

[18] Code3Arena, www.planetquake.com/code3arena/

[19] Gagnon, D. Videogames and Spatial Skills: An Exploratory Study. *Educational Communication and Technology*, *33*, *4* (1985), pp. 263-275.

[20] Leitheiser, B. and Munro, D. An Experimental Study of the Relationship Between Spatial Ability and the Learning of a Graphical User Interface, *Proceedings of the Inaugural Americas Conference on Information Systems*, Pittsburgh, PA, 25-27 August, 1995, pp. 122-124.

[21] Cockburn, A., and McKenzie, B., Evaluating the Effectiveness of Spatial Memory in 2D and 3D Physical and Virtual Environments, *Spatial Cognition*, *4*, *1* (April 2002), pp. 203-210.

[22] Marcus, A., Feng, L., and Maletic, J. I., 3D Representations for Software Visualization, *Proceedings of the 2003 ACM Symposium on Software Visualization*, San Diego, CA, 11-13 June, 2003, pp. 27-36.

[23] Feijs, L. M. G., de Jong, R., 3D Visualization of Software Architectures, *Communications of the ACM*, *41*, *12* (December 1998), pp. 73-78.

[24] Panas, T., Berrigan, R., Grundy, J., A 3D Metaphor for Software Production Visualization, *Proceedings of the Seventh International Conference on Information Visualization (IV'03)*, London, England, 16-18 July, 2003, p. 314.

[25] OpenSG Home, www.opensg.org.

[26] Doxygen, www.doxygen.org.

[27] Ed Chi, A Taxonomy of Visualization Techniques using the Data State Reference Model, Proceedings of the Symposium on Information Visualization (InfoVis '00), Salt Lake City, Utah, October 9-10, 2000, pp 69-75..

[28] Colin Ware, Ed Chi, Rich Gossweiler, Visual Perception for Data Visualization, Tutorial for Human Factors in Computing Systems Conference (CHI 2000), 2000, www-users.cs.umn.edu/~echi/tutorial/perception2000.

[29] Storey, M.-A., Wong, K. and Muller, H.A. How Do Program Understanding Tools Affect How Programmers Understand Programs, Proceedings of the Fourth Working Conference on Reverse Engineering, October 6-8 1997, IEEE CS Press.

[30] Lintern R., J. Michaud, M.-A. Storey and X. Wu, "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse", ACM Symposium on Software Visualization, (Softvis'2003), San Diego, pp. 47-56.

[31] Reiss, S.P. and Renieris, M. The BLOOM Software Visualization System, in Software Visualization - From Theory to Practice, MIT Press, 2003.