Bruce A. Maxwell, Nicolas Ward, and Frederick Heckel
Swarthmore College
Swarthmore, PA 19081
{maxwell, nward2, fheckel1}@swarthmore.edu

This paper describes the Swarthmore College entry in the 2004 Urban Search and Rescue [USAR] Event at the 2004 American Association for Artificial Intelligence Robot Competition. The primary focus of Swarthmore's entry was the development of a fast, responsive user interface for effectively tele-operating a robot with some autonomous capability. This interface was inspired by first-person video game interfaces, and enabled Swarthmore to place first in the 2004 USAR competition.

The Urban Search and Rescue [USAR] event at the American Association for Artificial Intelligence [AAAI] Robot Competition challenges robot teams to find victims in a simulated urban environment. In order to be successful at this task, the robot must answer three questions:

1. Where should it search?
2. How should it get there?
3. How does it identify victims?

The most successful teams so far answer these questions by providing a human operator with as much information as possible about the situation so that they can answer the questions accurately. Currently, the second question--regarding local navigation--is the only one of the three for which teams have attempted to give the robots autonomy.

It is our belief that the eventual solution to this problem will be a combination of effective user interfaces and robust autonomy on the part of the robot in answering the questions above. In other words, it is important to have a human in the loop, but the human should be processing and answering the most difficult questions encountered in the search, not providing input about basic search patterns or navigation except in difficult circumstances.

Since robust autonomous systems are still a difficult research problem, effective user interfaces are critical. Thus, the focus of our 2004 USR entry was to develop an effective, responsive user interface that permitted the operator to use a range of autonomy in navigation, and provide the greatest flexibility in the use of the robot's hardware. As a result of the effective interface and the robot's semi-autonomous navigational capability, Swarthmore placed first in the 2004 competition.

.Copyright 2004. American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

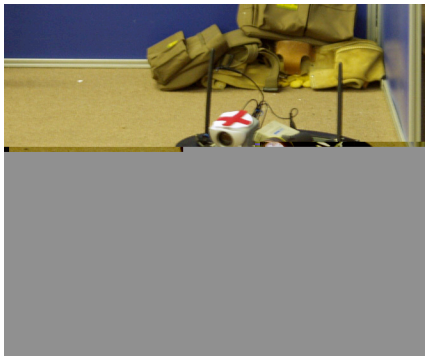

Figure 1: One of the Magellans entering the USR arena.

Swarthmore's robots in the USR competition were two iRobot/RWI Magellan Pro's with onboard 850MHz Pentium III computers running Linux. Figure 1 shows one of the robots entering the USR arena. The Magellans come with 16 sensor panels, each containing a bump sensor, an IR sensor, and a sonar. In addition, each Magellan has a Canon pan-tilt-zoom [PTZ] camera mounted on top, near the front, with a pan range of -100º to 100º, a tilt range of -30º to 90º, and a 16X optical zoom. This year, one of the cameras was a VC-C4, the other a VC-C50i with infrared night-mode capability for enhanced vision in dark areas. The camera video is connected to a bt848-based PCI framegrabber, and it receives pan-tilt-zoom commands via the serial port.

Because the robots are wheeled, and do not have a high clearance, they are only able to enter the yellow area of the arena, or parts of the orange that are clear of debris more than 2cm high. So the focus of the interface is on making the operator more productive, providing a large quantity of accurate information about the environment, and enabling the robot to traverse the area safely and quickly.

The software used locally on the robot to manage the robot's navigation and sensors in the 2003 contest was based on the REAPER concept (Maxwell *et. al.* 2001). Since then, we have upgraded the vision system (Maxwell *et. al.* 2003), switched to using the Inter-Process Communication [IPC] package for all communication between processes (Simmons and James, 2001), and this year upgraded the navigation component and added a software monitoring program for managing the various software modules.

The primary focus of our work for the 2004 contest was the interface for managing the robot during the Urban Search and Rescue [USR] task.

Early in the course of this design project, it was realized that the greatest advances in user interfaces [UI] in the past ten years have been made in the realm of video games. Computer and console games alike must have clear interfaces which provide the player with both controls and informational displays that allow for very fast response times to situations in a dynamic environment. These interfaces have developed at an incredibly rapid pace, with perhaps the pinnacle of UI design being that of so-called First-Person Shooter [FPS] games (Unreal Tournament, 1999). FPS games are the most appropriate interface paradigm for a project such as USR, as they represent a three dimensional world with which the player can interact. Navigating a three-dimensional game world in search of computer-generated enemies is analogous to the task of finding injured and/or trapped victims in a USR scenario. Mapping that three-dimensional environment down to a two-dimensional computer display is simple enough using a live video feed, but making it intelligible, informative, and useful for the robot operator is far more difficult.

The most successful FPS games have very simple interfaces: the main portion of the interface is dedicated to a large viewing area that contains the game world. Small status areas around the perimeter are allocated for information regarding the player's health and equipment. Most games also have a radar mini-map showing the direction to enemies or other players, as well as a separate screen containing a map of the current game level. The interface we ultimately used followed this format closely. However, the design of the interface module permits simple rearrangement of the visual items on the screen to enable testing different interface configurations and looks.

The interface's viewable layout and internal structure are defined by the relationships between a small set of configured objects, consisting of a *view*, *viewports*, *visualizers*, and *widgets*. The parameters for each type of object are specified in a configuration file, which allows the interface to be quickly reconfigured at run-time. The arrangement of these objects in the interface's view tree determines how user interaction events and messages broadcast by one of the robots are handled. This philosophy of highly generic code is maintained throughout the interface, which allows for operational flexibility as well as easy expansion of functionality in future updates to the interface's capabilities.

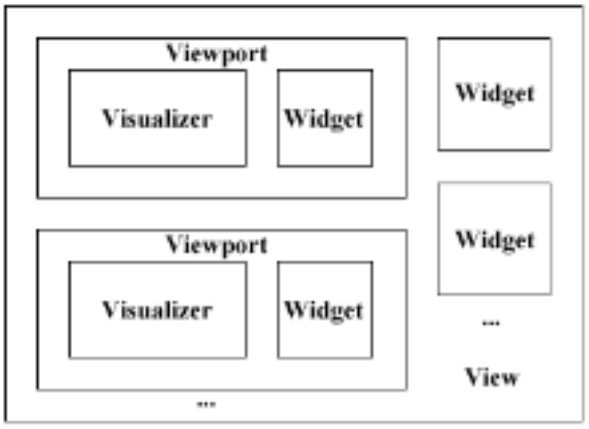

Figure 2 Relationship of interface screen elements.

As show in Figure 2, the *view* defines the screen for the entire interface, as well as global parameters that apply to all viewports. The view contains some number of configured viewports and widgets, determined by the contents of the configuration file. The global view widgets are always active, and their scope is the entire interface. An example global view widget would be the keyboard shortcut for quitting the interface. No widgets are hard-coded, so even a widget as basic as quit must be specified in the configuration file.

A *viewport* is the display for a single visualizer, so each viewport provides some sort of information to the user. A given viewport has a defined position and dimension in the on-screen view, and each is associated with a single visualizer. A given viewport's visibility can be toggled, so the information currently displayed in the view is contextual and dependent on the current state of the robot and/or the interface. Each viewport contains a single visualizer for converting some information into a displayable format, as well as some number of widgets. These local viewport widgets are only active when the viewport is visible, and their scope is confined to the viewport.

A *visualizer* converts incoming data from a robot into a visual representation that is meaningful to the user, such as text output, an image, a status icon, or a graph. Each visualizer is associated with a message broadcast by a module on one of the configured robots. Multiple visualizers can process a given message, for example if the message contains different data types that need to be displayed separately. When a message is received, it is processed by all of the visualizers in visible viewports that are associated with the message's originating module and robot.

A *widget* responds to an input event from the user, such as a mouse click, key-press, or joystick movement, and

takes action based on that input. The action taken could be an internal status change, or a message that is generated and passed to a robot. Widgets that respond to a mouse click have a defined area within their parent view or viewport.

Nearly every aspect of the interface is defined in a single XML configuration file (Goldfarb, 2003). This includes the network connection information and metadata for each robot, the modules that should be started on each robot, the configurations for external USB controllers, such as joysticks or game-pads, as well as the layout of viewports within the view and the message and control bindings for visualizers and widgets.

While allowing all of these interface parameters to be configurable makes the configuration file large, it allows each user to have their own custom (and potentially completely different) version of the interface. This is important from an HRI perspective, because different users may have different preferred interface layouts. Also, the interface can be quickly reconfigured to run on different computers with different available control peripherals. Since changing configurations is just a matter of restarting the interface with a different configuration file, it would be most efficient to have several configurations available to swap in as needed.

The XML format was selected because it is an open format that is easily human readable and familiar to many people due to its wide use and similarity to HTML. In addition, easily obtained open source XML parsing libraries made it easy to make the configuration process an important part of the interface. In fact, the parsing and setup code makes up over 20% of the entire code for the interface software.

The example interface configuration in Figure 3 was used by an operator during the AAAI 2004 USR competition. It features a centered video feed, emphasizing the primary method by which most users receive their situational awareness. Superimposed on the camera image are the pan-tilt-zoom indicators for the camera, which prevents the user from confusing straight ahead in the image with straight ahead on the robot. The two displays in the lower right of the view are the range sensors, which show obstacles within a meter of the cylindrical robot that was used, and the map that is generated over the course of a run. The interface is run full-screen to avoid background clutter that might distract the operator. The green bars in the video display indicate half-meter distances on the ground place, with the first bar a meter away from the camera lens. These provide necessary scale information since the viewpoint of the robot is substantially different from that of an adult human. The user can switch between robots by using the space bar on a keyboard, which switches all of the visualizers.

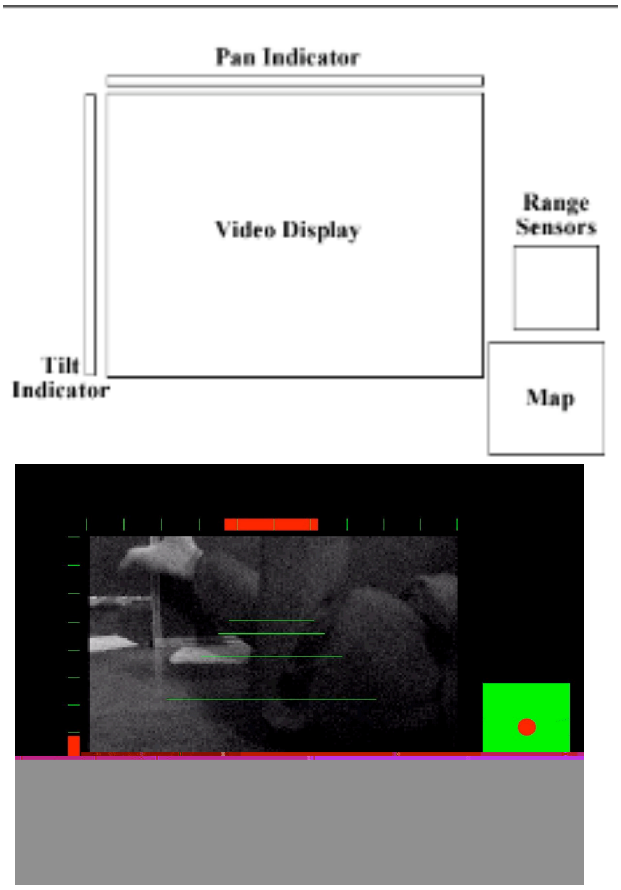

Figure 3 top) Description of screen items, bottom) screen shot from the 2004 AAAI Robot Rescue Competition.

Figure 4 shows the previous Swarthmore interface used in the 2003 competition. The differences between the two versions of the interface are obvious at first glance. The old version--Figure 4--used a small window, with many on-screen controls that had to be used with the mouse. There was a set of keyboard bindings that could be used more efficiently by an experienced user, but the learning curve was steep, making the interface difficult to use for a beginner.

In contrast, on the new interface almost all of the controls are on a single joystick, a more intuitive control method used in many video games and in many real-world control applications. Furthermore, in the new interface all of the information needed by an operator is shown together in a single unified display instead of in separate windows.

The impetus for these changes came out of the results of watching inexperienced users attempting to drive the robots using the old interface, which highlighted its complexity. The significant improvements in this version have enhanced the ability of a single operator--novice or experienced--to remotely control one or two robots in a serial fashion.

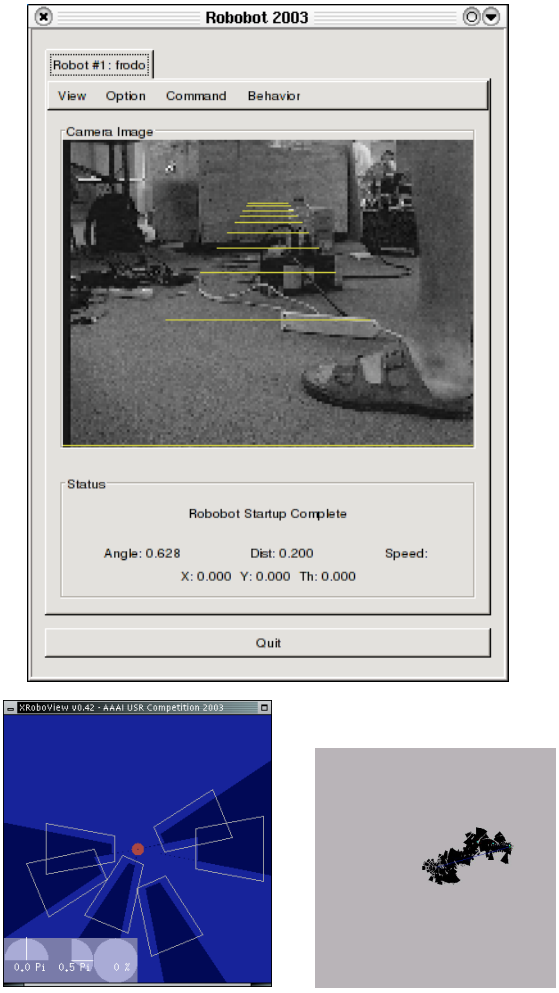

Figure 4 Interface windows for the 2003 version. Each robot used three windows to display the information.

In addition to changes in the interface, we upgraded the robot's navigational and visual capabilities for 2004. We had two goals in making the upgrades, 1) to make control of the robot smoother and more natural, while maintaining the robot's ability to react autonomously to perceived obstacles, and 2) to speed up the frame rate of the video stream coming from the camera.

The vision system for the robots is the Swarthmore Vision Module [SVM] (Maxwell, Fairfield, *et. al.*, 2003). In order to speed up the frame rate of the video we made two changes to the robot system, one involving software, one involving hardware.

For the hardware upgrade we switched from 802.11, to 802.11b wireless, which increased the network bandwidth from 1Mbps to 5Mbps (in actual practice). However, a medium size (320 x 240) color image is 1.8 Mbits, which means that the maximum frame rate even on the new hardware is less than 3fps, which is insufficient for driving quickly even in a fairly open environment.

In order to reduce the frame rate we integrated a sliding level of image quality into SVM. When requesting an image from SVM, the interface can request any one of 12 different levels of image quality, depending upon the situation. We obtain the twelve levels of image quality by using three different levels of compression {none, medium, high} two image sizes {small, medium}, and two image types {color, greyscale}.

While driving, a small (160 x 120) greyscale, highly compressed image is sufficient to avoid obstacles and packs each image into less than 80kbits, enabling a frame rate of better than 5Hz with minimal lag. When searching an area for victims, a medium size, medium compressed image, at 1.2Mbits per image, is a good compromise between frame rate and frame quality.

The compression is based on run-length encoding after reducing the number of bits per color band to increase the average length of runs. The compression scheme is guaranteed to reduce the size of the original image.

For greyscale images, the original full-color image is first scaled to the appropriate size and converted to greyscale by grabbing just the green band, which is the most sensitive (hence, least noise) color band in most CCD cameras, and it does not involve a linear transformation or averaging of the color bands. The medium level of compression then represents a run using 8-bits per run, with 2 bits of run length information and 6 bits of intensity information. In the worst case--all runs are of length 1--the image does not compress at all and reduces to 6 bits of intensity information per pixel. In the best case--all runs are of length 4--the image size is reduced by a factor of four. Typical performance was reduction by a factor < 2.

The low level of compression uses 3 bits of run-length information and 5 bits of intensity information per run, with a potential best-case reduction by a factor of 8. Typical performance was reduction by a factor of about 2 because of the relatively complex scenery in the robot arena. The compression algorithm can compress the image in place in a single pass, simplifying memory requirements and minimizing the computational load on the robots.

For color images, SVM uses a similar scheme. The medium level compression replaces every 24-bit pixel (8-bits per color channel) with a 16-bit pixel using a 5/6/5 bit representation for red/green/blue. with no RLE compression. The low level compression uses a 4/4/4 red/green/blue representation and uses 4 bits to encode run-length. Like

the greyscale images, the algorithm works in place in a single pass.

It is likely we could achieve higher compression rates by further reducing the number of buckets per pixel, which would result in longer runs. However, the complex nature of the images severely limited the average length of the runs even with reduction to 16 or 32 buckets per color band/ intensity. We chose not to implement more complex compression methods because we could not devote more CPU time to compression on-board the robots, and the RLE compression works in place in a single pass.

The current implementation provides a 12-step sliding compression scale. While using the more highly compressed video stream the limitation on the frame rate is not the compression time or network bandwidth, but the display time required by the user-interface to scale and place the uncompressed video image on the full-screen display.

The Navigation module is based upon velocity space, using the dynamic window approach (Fox *et al*, 1997). As our major concern with the competition this year was not with autonomy, but rather human control, we found it more useful to focus on finding safe preferred velocities rather than achieving goals.

The original velocity space implementation used three parameters to manage the speed and clearances within the system: alpha, beta, and speed. The alpha parameter scaled the progress towards the goal, the beta parameter scaled the clearance factor, and the speed parameter scaled the speed score (Fox *et al*, 1997). Because of our focus on safe trajectories rather than goal achievement, we were able to reduce the parameters of the velocity space to two.

The first parameter is a general sensor growth parameter applied to each sensor before processing the velocity space. One advantage to a round robot is that a single growth parameter is often as effective as different values for each sensor--though we are also experimenting with a two parameter elliptical growth approach which would be more appropriate for moving through tight doorways. The second parameter provides the navigation module with some flexibility, defining how far off the preferred heading the robot is allowed to move, enabling it to choose a course around an obstacle in its path rather than merely stopping. This effectively creates a dynamic window by defining a limited portion of the velocity space which we are willing to examine.

The value used for growth in the competition was 1.4--the sensor grown by 40% of the robot's radius--a value at which arrived after empirical testing. Outside of a cluttered environment such as USAR, a larger value such as 1.7 is more appropriate, especially in environments with moving obstacles such as people.

The second parameter is "autonomy", which is a real value in the 0-1.0 range. During velocity selection, this value is used to scale possible velocities. In the case of 0, the rotational velocity is limited to that requested by the control interface, and the navigation module will move as quickly as possible. As autonomy increases to one, the range of rotational velocities that will be considered by the module increases. The module will still prefer velocities that are closer to the requested velocity, by scaling the velocity score (based on distance until a collision along the path) according to its distance from the requested velocity. The scaling S_A factor is

$$S_a = 1.0 - \left(\frac{v_c - v_r}{d_{\max}}\right) \quad (1)$$

where v_c is the current velocity, v_r is the requested velocity, and $d_{\max}$ is the maximum allowed velocity distance (how far the robot can travel safely at that velocity).

We call it the autonomy parameter because, as it increases, the robot will take more control over the path it follows. When directly tele-operating the robot, it should be set to 0 to provide maximum human control. When using shared or fully autonomous control, it should be set to a higher value. When compared to the original velocity space method, this parameter approximates "alpha".

For the USAR competition, we used a value of zero to provide maximum operator control and because the current sensor growth implementation is not yet sufficiently tested.

When developing long-term robot systems, it is imperative to use a modular architecture for the robot systems. This enables us to reuse code and develop new capabilities quickly--an important feature with a small number of undergraduate researchers working in a limited time-frame. We have divided our robot systems into a number of different module domains which fall along fairly natural divisions of labor: vision, navigation, mapping, control, and communication. For the 2004 competition, we were able to completely rewrite our navigation and control modules with only very minor changes to vision and mapping.

The major disadvantage to using a modular architecture composed of multiple threads is that sharing information between different sensor domains becomes much more difficult as they do not necessarily share access to the same memory structures. To solve this problem, we make use of Carnegie Mellon University's Inter-Process Communication system (Simmons and James, 2001). CMU IPC allows us to easily pass even very complex data structures over TCP/IP within robots, between robots, and to other connected systems all within the same software framework.

As the robots use multiple modules, sometimes different sets of modules for different tasks, module management is a major concern in this sort of system. We addressed this problem, and that of consistent communication between modules, by making two major upgrades to our system.

The two major new additions to the software architecture in 2004 are GCM and Robomon. The General Communications Module [GCM] is a message framework designed to be a centrally defined API for robot control. On the most basic level, it is little more than a set of message definitions for use with CMU IPC, and a collection of useful common utilities that each robot module needs. Considered from a more abstract standpoint, GCM actually defines the capabilities of our modules by defining the messages and the expected reaction to each message. A navigation module is expected to implement actions to respond to "move", "achieve", and "override" messages, while a vision module is expected to be capable of providing images of the world on demand. GCM has further streamlined the coding and design process by defining the required behavior for each type of action, thereby speeding integration.

Robomon is a separate module--run as a daemon--which runs on each robot to insure module availability and simplify robot startup. In previous years, it has been necessary to open multiple windows to start each robot module and monitor its current state. This year we took a different approach by creating a monitor program which starts when the robot boots up. Robomon is defined by an XML file containing information about each different module available to the system. When the outside control module connects to the communications server on the robot, it can then tell Robomon what modules it requires. Each module will then be started and monitored. In the case of a module crash, Robomon will restart the module; this feature was of critical importance in our final USAR runs, when a bug emerged that caused our mapping module to crash repeatedly. While this did impact our ability to create consistent, complete maps, it did not impact the overall system as Robomon would restart the map module, making it possible to continue using the robots effectively.

The robots performed well in the USAR task, scoring over 7 points per round, on average, and having no rounds with a score of zero or less. The robots drove safely in the arena at speeds higher than we have been able to achieve previously because of both the velocity space navigation and autonomous reactivity and the faster frame rate due to the video compression. The interface also provided good situational awareness to the operator, and, unlike previous years, there were no significant operator errors--such as driving with the camera sideways--due to lack of information about the robot's environment or status. Because of these improvements we were able to place first in the 2004 USAR competition at AAAI.

The primary future work on the robot systems is improved mapping and implementation of a localization scheme based either on visual input or using some form of SLAM with the sonar and IR data (Montemerlo *et al*, 2003). An improved map, and good localization will make it possible to manage multiple robots simultaneously, and to give the robots the ability to undertake more autonomous search with less human oversight.

This work was supported in part by NSF IIS-0308186, and the American Association for Artificial Intelligence.

[1] Dieter Fox, Wolfram Burgard, and Sebastian Thrun, "The Dynamic Window Approach to Collision Avoidance", *IEEE Robotics & Automation Magazine*, 4(1), March 1997.

[2] C. Goldfarb, *XML Handbook, 5th ed*, Prentice-Hall, 2003.

[3] T. W. Malone, "Heuristics for Designing Enjoyable User Interfaces: Lessons from Computer Games", In *Proc. of Conf. on Human Factors and Computing Systems*, pp 63 - 68, 1982.

[4] B. A. Maxwell, L. A. Meeden, N. S. Addo, P. Dickson, N. Fairfield, N. Johnson, E. G. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter, E. Silk, 2001, "REAPER: A Reflexive Architecture for Perceptive Agents", *AI Magazine*, American Association for Artificial Intelligence, 22(1): 53-66.

[5] B. A. Maxwell, N. Fairfield, N. Johnson, P. Malla, P. Dickson, S. Kim, S. Wojtkowski, T. Stepleton, "A Real-Time Vision Module for Interactive Perceptual Agents", *Machine Vision and Applications*, 14, pp. 72-82, 2003.

[6] B. A. Maxwell, N. Ward, and F. Heckel, "A Human-Robot Interface for Urban Search and Rescue", in *Proc. of AAAI Mobile Robot Competition and Exhibition Workshop*, Acapulco, pp. 7-12, August 2003.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges", in *Proc. of IJCAI 2003*, Acapulco, MX, 2003.

[8] Reid Simmons and Dale James, *Inter-Process Communication: A Reference Manual*, Carnegie Mellon University, March 2001.

[9] *Unreal Tournament*, Epic Games Inc, 1999.