

## **Building Cost-effective Research Platforms: Utilising Free | Open-source Software in Research Projects**

Tony Meyer

*Institute of Information and Mathematical Sciences,*

*Massey University, Auckland, New Zealand.*

*T.A.Meyer@massey.ac.nz*

### **Abstract**

When prototyping or developing a system for use in research work, it is often necessary to create an entire system, even if only one part of the system is the focus of the research. Free | open-source software offers a solution to this problem, allowing the creation of cost-effective research platforms, utilising peer-reviewed, rapidly-developed code that is easily modified. One form of free | open-source software that is regularly used in research projects is engines from 3d games such as Unreal Tournament (Lewis & Jacobson, 2002). Although the core rendering engine is proprietary, the game engine is able to be freely utilised as a reasonably generic rendering engine and physics simulator, and most of the game code is modifiable.

A synthetic characters development project, outlined in this paper, uses the Unreal Tournament game engine, via the Gamebots socket tool, as its primary output system. Several other free | open-source software packages are utilised, including a speech recognition system (Sphinx from Carnegie Mellon University), a speech generation system (Festival-Light from Carnegie Mellon University), and a vision system built with Lego™ Mindstorms™ (and the open-source NQC), cheap web-cameras and Intel®'s OpenCV library. These modules all communicate via standard network sockets and are able to operate independently. Each module required a different level of modification in order to form part of the synthetic actors system, from no modification at all (Unreal Tournament), to light modifications (Festival-Light), to a new system based on open-source code (the NQC-based tripod code); one package – the OpenCV library – is simply linked to by completely new code.

### **Research and Free | Open-source Software**

Often research does not stand-alone; in order to develop one part of a system it is often necessary to have access to other parts of the system. However, developing the systems peripheral to the central area of interest steals time and other resources from the core research, and often merely duplicates work already done by others. Essentially, “most complex, dynamic ... research environments require considerable efforts to build and maintain, and therefore, there is a general scarcity of such infrastructures available for research use” (Adobbati et al., 2001). Pre-built systems can be used, but these are often expensive and may end up consuming time when it is necessary to code around quirks – or worse, bugs – in that software. Free and open source software offers a solution to these problems.

### **Free | Open Source Software**

There are a number of different definitions of the term ‘open-source software’, ranging from simply any software where access to the source code is provided, to software that abides by the license set out by the Open Source Initiative (Open Source Organisation, 2003). In this paper, ‘free | open-source software’ (FLOSS) refers to any software that may be freely<sup>1</sup> used, and that provides access to part or all of the

---

<sup>1</sup> FLOSS literature frequently refers to the colloquial terms “free speech” and “free beer”. In terms of cost-effectiveness, we are most concerned about “free beer” (i.e. the software, or parts of it, can be used without charge and without extensive restrictions). However, as outlined below, the “free speech”

source code. The benefit of open-source software is that “[it] promotes software reliability and quality by supporting independent peer review and rapid evolution of source code” (Open Source Organisation, 2003). Given that independent peer review is central to the research process, and that rapid development of source code is desirable (moving time from development to research), open-source software meshes well with the research process.

Other than the opportunity for independent peer review, and assistance with development, there are other advantages to utilising FLOSS in research projects. Some are pragmatic – given that funding for any project is limited, the opportunity to evaluate a number of different packages, and then utilise one, without any financial cost, is highly beneficial. In addition, although it is preferable to use the FLOSS package without any modifications, if these are necessary (if a bug is found, or if a modification allows better linkage with the core research), this is a simple matter.

### **Using Free / Open Source Software in the Development of Synthetic Actors**

The author is involved in a project where the aim is the development of a synthetic character able to take part in rehearsals for a stage performance (Meyer & Messom, 2002). Although the key development required is an artificial intelligence or planning engine, a full computer-theatre system is required, in order to fully test the system. This consists of, at least, a graphical output system (an actuator), and some form of sensor (such as a vision system or an audio recognition system), and for this particular project includes semi-realistic graphical output, speech synthesis, speech recognition, and a multi-camera vision system. Rather than spending time developing these systems, peripheral to the core research interest, this project utilises a number of freely available open source systems.

### **Game Engines**

Although there is a widely held stigma of violence associated with first-person games, “there has been a long history of unpublicised cooperation between computer scientists and the game industry” (Lewis & Jacobson, 2002). The game industry is extremely well funded, and this allows development of very high quality game engines; today, these engines are often available for members of the public (or research institutions) to use – the two most well known examples are Quake<sup>2</sup> (and its successors) and the Unreal series<sup>3</sup>. The companies that developed these games, while hiding the core of the game engine (in some cases), have developed APIs that allow users to take advantage of the game engine.

Here, we refer to the ‘game engine’ as “that collection of modules of simulation code that [does] not directly specify the game’s behaviour (game logic) or game’s environment (level data). The engine includes modules handling input, output (3D rendering, 2D drawing, sound), and generic physics/dynamics for game worlds” (Lewis & Jacobson, 2002). For research that requires only a limited level of performance (for example faithful maintenance of player, object and terrain locations), game engines are able to provide “superior platforms for rendering multiple views and coordinating real and simulated scenes as well as supporting multi-user interaction” (Lewis & Jacobson, 2002). Projects directly concerned with input, output or physics and dynamics are probably better off developing their own systems, but those that merely wish to use such systems are able to avoid the costs of doing so.

### **Machinima, Soar, Esc, and Gamebots**

While the primary purpose of allowing access to the game engine is to allow users of the game to develop ‘mods’ (modifications to the game, whether levels, characters, weapons or otherwise), projects in a wide range of fields have availed themselves of this access. Outside of computer-science, game engines form the basis of the art form Machinima – “the convergence of three creative mediums: filmmaking,

---

element is also important, as it allows the modifications that are often necessary to integrate FLOSS into a research project.

<sup>2</sup> From id software – <http://www.idsoftware.com>

<sup>3</sup> From Epic – <http://www.unreal.com>

animation and game development” (Academy of Machinima Arts & Sciences, 2003a). “By combining the techniques of filmmaking, the flexibility of animation production and the technology of real-time 3D game engines, Machinima makes for a very cost- and time-efficient way to produce films” (Academy of Machinima Arts & Sciences, 2003b).

There are many examples of research (and other) projects that make use of game engines, particularly the Quake, Unreal Tournament and Half-life<sup>4</sup> families. These range from projects that operate entirely within the game, to those that only use the central engine itself. An example of research that utilises game engines without any alteration, and also perhaps the most well-known example and certainly one of the earliest examples is the University of Michigan’s Soar project (Laird, Newell, & Rosenbloom, 1987). As part of the Soar project, research in planning used Quake and Unreal ‘bots’<sup>5</sup>, attempting to outplay (by ‘outthinking’, rather than through raw speed) human players of the games.

Also using the Unreal (Tournament) game engine is Esc (Martin, 2002), an environment developed to research believable agents. Esc, however, replaces almost every visible element of the Unreal Tournament game (with those appropriate to a nightclub environment), only utilising the core game engine itself. Like many projects that use FLOSS software, the Esc project itself is available, including all source-code, in the hope that others researching believable agents will find it useful. The University of Southern California’s Gamebots project is “an infrastructure for multi-agent systems research that supports different platforms” (Adobbati et al., 2001) and is similarly available. Gamebots allows control of Unreal Tournament ‘bots’ via network sockets, abstracting control of the ‘bot’ from the game, and simplifying development in languages other than Unreal’s internal scripting language.

### Synthetic-actors project

The Unreal Tournament engine (via the Gamebots package) is used in the synthetic-actors system that the author is currently developing (Meyer, 2003); with the exception of the central core, all the components of the system project are free | open-source software packages. Taking advantage of the modularity of the individual components, each section of the synthetic-actors system (other than the core) acts independently. Communication between the modules is accomplished via standard network sockets,



Figure 1: Unreal Tournament Game Engine Output

<sup>4</sup> From Valve software – <http://www.half-life.com>

<sup>5</sup> In computer games, a ‘bot’ is a player that is not human-controlled.

allowing the system to operate on one machine (where each module has a localhost address), or to take advantage of multiple machines if they are available.

The core of the system is currently an interval engine (Pinhanez, 1999) that processes a specified interval script, which details the time structure of the performance. The core is the only component of the system expected to undergo extensive changes (as it is the focus of the research). The open-source nature of the other modules allows the core to change even in ways that do not fit with the other modules, as modifications are able to be made without any difficulty. An aim of the synthetic actors system is that it is highly modular and easily integrated into other systems. As part of satisfying this aim, interval scripts are written in XML (Meyer, 2002) rather than a proprietary format (as in the original (Pinhanez & Bobick, 2002)). Another advantage of using an open-source specification is that open-source software is available to process it – in this case the open-source library Expat<sup>6</sup> is used for XML processing.

### Unreal Tournament

The Unreal Tournament game engine acts as both a sensor (of the virtual world) and actuator (both audio and visual – see figure 1). All communication between the core and Unreal Tournament is passed through the Gamebots system, which is packaged as a ‘mod’ (plugin) for the game. The core communicates with the Gamebots system via two network sockets (see figure 2) – one from the core, which carries instructions to the graphics system (“move to here”, for example), and one to the core, which carries information about what the actors can ‘see’ and ‘hear’ in the virtual world.

The Gamebots system required only minor modifications in order to be used in the synthetic-actors system; mostly these consisted of additions to send additional information about the virtual world to the core. Unreal Tournament itself required no modifications (other than those that the Gamebots system makes), although new ‘skins’ and models for characters and objects are required, unless performances take place within the settings provided by Unreal Tournament (which tend towards violence). Another advantage of the free | open-source nature of the game engine is that many such models are freely available, both from research projects (such as Esc (Martin, 2002)), and from game players themselves.

### Free Speech

Even more time-consuming than developing a game engine is development of speech recognition and generation packages. The synthetic-actors system utilises two packages jointly developed by Carnegie Mellon University and Edinborough University, cmusphinx (for recognition) (Huang, Alleva, Hon,

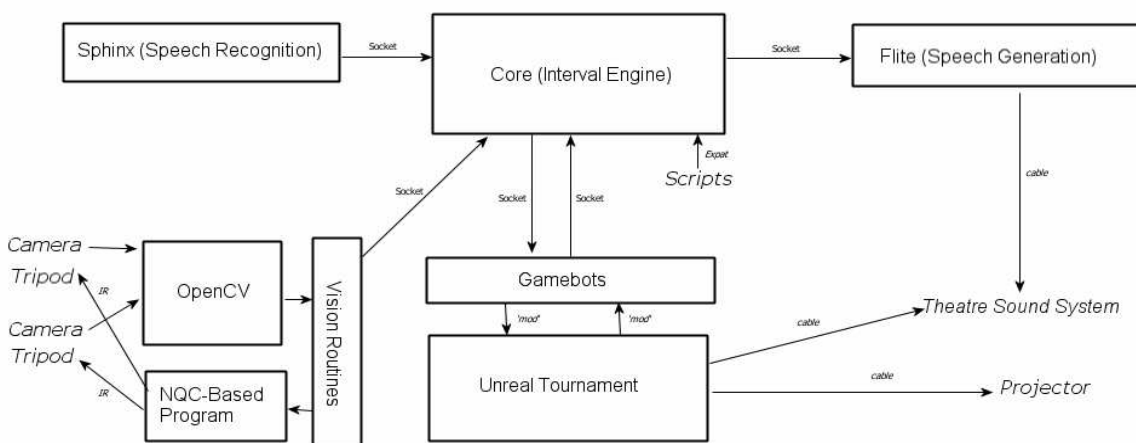


Figure 2: Synthetic actors system

<sup>6</sup> <http://expat.sourceforge.net/>

Hwang, & Rosenfeld, 1992) and flite (for generation). Cmusphinx<sup>7</sup> comprises three complex speech recognition systems: Sphinx-2 is the least accurate system, but operates in real time, whereas Sphinx-3 aims more towards accuracy (although a fast, 'light' version is available); Sphinx-4 is a port to Java, designed to be more flexible than the previous systems. Although the synthetic-actors system currently uses the fast version of Sphinx-3, either of the other systems could replace this if desired.

The two universities have also developed speech generation software – primarily the Festival (Taylor, Black, & Caley, 1998) suite of tools, and also the Festival-Lite (flite) system. Flite is designed to be portable, fast and light on resource requirements, and as such is ideal for forming part of a computer-theatre system. In addition to the resources that are saved by not having to develop speech systems, the use of open-source systems such as these provides an additional advantage – the theoretical base has been extensively reviewed. Unlike the Unreal Tournament system, the speech tools are provided by a research institution, have been developed as research tools and prototypes, and many published papers detail the processes that the systems use.

### Free | open-source software solutions for proprietary hardware

The Lego™ Mindstorms™ hardware is a cost-effective method of building simple computer-controlled robotic systems (for example (Levesque & Pagnucco, 2000), (Kumarl, 2000), (Lund & Pagliarini, 1999)). The vision system for the synthetic-actors project required two tripods, able to pan left and right (see figure 3). Although such tripods are commercially available, a Lego™ solution offered both cost-effectiveness and adaptability that commercial tripods could not match. Software to control the hardware is provided with the Lego™ Mindstorms™ package; however this (proprietary) software is designed for children, is platform-specific, and is large and slow. There are FLOSS alternatives to this proprietary



Figure 3: Lego™ Tripod with QuickCam® Express® camera

---

<sup>7</sup> <http://cmusphinx.sourceforge.net>

software, however, that work flawlessly with the Lego™ hardware.

Two modules were necessary for controlling the tripods; the first is executed on a standard computer and sends infra-red messages to the RCX, while the second is downloaded (once) to the Lego™ RCX™<sup>8</sup> and controls movement of the motors based on the messages received. As the program that executes on the RCX™ is fairly simple, the proprietary Lego™ software (an integrated development environment, complete with a graphical ‘block’ programming language) was used to code and download the program. The program responsible for sending infra-red messages, however, could not be coded with the standard Lego™ system, which is designed only for designing and transmitting programs to execute on the RCX™.

Although there are several FLOSS packages that work with the Mindstorms™ system, ranging from Java to Visual Basic, the most commonly used is NQC (Not-Quite-C), originally developed by Dave Baum<sup>9</sup>, which provides a C-like programming language for accessing the Mindstorms™ hardware. NQC is a programming language (syntactically similar to C) and a compiler which can also transmit infra-red commands to a RCX™. Unlike the other FLOSS packages used in the synthetic-actors system, the NQC software was not directly included in the synthetic actors system; although this was the original implementation, the full NQC system was too slow to send rapid infra-red commands.

Like most open-source projects, the license for NQC (v1.0 of the Mozilla Public License), allows re-use of portions of the code, as long as this is acknowledged, and subject to certain other conditions. Taking advantage of this ability a new module for the synthetic-actors system was developed, based on (and including portions of) the NQC code, but considerably trimmed down to focus on the transmittal of infra-red messages to the RCX™. Like the other modules, this receives commands via a network socket; it then sends the appropriate data to the infra-red tower, which transmits this to the Lego™ RCX™.

## Vision

In the synthetic-actors project, the actors are expected to interact with not only other virtual objects, but also interact with human actors on stage. Although some sensory information is provided via the speech recognition system, the majority of sensory information (about the real world) is provided via a vision

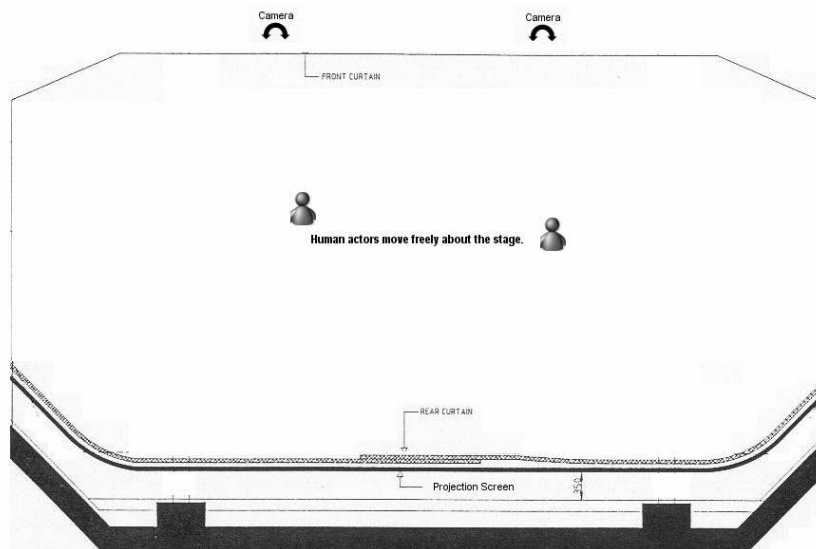


Figure 4: Stage layout indicating camera and projector screen positions.

<sup>8</sup> The RCX™ is a Lego™ ‘brick’ that essentially holds a miniature computer with 32K of RAM and capable of running up to five small (user-defined) programs.

system. Intel Research has developed the Open Computer Vision Library (OpenCV), “intended for use, incorporation and modification by researchers, commercial software developers, [and] government and camera vendors” (Intel Corporation, 2003). Unlike the other FLOSS modules in the synthetic-actors system, OpenCV is not a stand-alone package, but a library of routines to use.

The vision module of the synthetic-actors system pulls together different elements of OpenCV. The OpenCV routines provide basic frame capturing from the cameras (two Logitech® QuickCam® Express web cams, see figure 3), which are placed at the front of the stage (see figure 4). Although other methods could be used to perform this capturing (Microsoft™’s DirectX™, for example), the open-source and cross-platform nature of OpenCV is appealing. Simple image processing is then performed on the captured frames, and information is sent to the core about what the vision system has determined. The two primary processes that are currently carried out are motion tracking and object detection. Rapid object detection is performed via a cascade of boosted classifiers based on Haar-like features (Kuranov, Leinhardt, & Pisarevsky, 2002), while the motion tracking is based on simple silhouette analysis.

## Building Cost Effective Research Platforms

Several free | open-source packages have been utilised in the development of a synthetic actors system. While the details of implementation (calling a library, reusing portions of code, or using untouched) differ between the systems, in all cases the packages are able to form part of the larger system as a result of being free | open-source software. In order to put together a cost-effective synthetic actors research platform the modules forming the system need to be either free or extremely cheap. Only two of the modules discussed are not free – Lego™ and Unreal Tournament; both, however, are cheap, and are able to be easily integrated into the overall system by other free | open-source packages. In the case of Unreal Tournament, this integration is facilitated by the open-source nature of the majority of the code.

Some of the modules have been developed by other research groups (Sphinx, Flite and Gamebots), while others have been developed by commercial organisations (Unreal Tournament, OpenCV), and others by enthusiasts (Expat, NQC). Whoever the developer, all the systems have undergone a form of peer-review in their use (modified or not) by other users. Although commercial systems are also used by a wide range of groups, the underlying code is not available for examination as with free | open-source systems and this limits not only the level of review, but also the possibility for implementing solutions to flaws in the systems.

## Conclusion

Having access to reliable, modifiable, cost-effective solutions to the required systems has enabled much more rapid development of the synthetic actors system. Without the free | open-source systems discussed, the system would not be as full-featured, would be less reliable, and would have required a substantially greater time commitment to develop. The use of free | open-source software has enabled both financial and time resources to be diverted from development of system components that are of secondary interest (but none-the-less required) to the central core of research interest. The synthetic actors project is not unique in this regard, and it is likely that for the development of many research platforms free | open-source software can provide a cost-effective, rapid, solution.

## References

Academy of Machinima Arts & Sciences. (2003a). *Academy of Machinima Arts & Sciences* [Website].

Retrieved 1 April, 2003, from the World Wide Web: <http://www.machinima.org>

---

<sup>9</sup> <http://www.baumfamily.org/nqc>

- Academy of Machinima Arts & Sciences. (2003b). *What is Machinima* [Website]. Retrieved 2 April, 2003, from the World Wide Web: <http://www.machinima.org/whatis.html>
- Adobbati, R., Marshall, A. N., Scholer, A., Tejada, S., Kaminka, G. A., Schaffer, S., & Sollitto, C. (2001). *Gamebots: A 3D Virtual World Test-Bed for Multi-Agent Research*. Paper presented at the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada.
- Huang, X., Alleva, F., Hon, W., Hwang, M., & Rosenfeld, R. (1992). *The SPHINX-II Speech Recognition System: An Overview* (CMU Technical Report CMU-CS-92-112): Carnegie Mellon University.
- Intel Corporation. (2003). *Intel Research* [Website]. Intel. Retrieved 8 April, 2003, from the World Wide Web: <http://www.intel.com/research/mrl/research/opencv/>
- Kumar, A. (2000). *Using Robots in an Undergraduate Artificial Intelligence Course: An Experience Report*. Paper presented at the 31st ASEE/IEEE Frontiers in Education Conference.
- Kuranov, A., Leinhardt, R., & Pisarevsky, V. (2002). *An Empirical Analysis of Boosting Algorithms for Rapid Objects With an Extended Set of Haar-like Features* (Intel Technical Report MRL-TR-July02-01).
- Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar - an Architecture for General Intelligence. *Artificial Intelligence*, 33(1), 1-64.
- Levesque, H. J., & Pagnucco, M. (2000, August). *Legolog: Inexpensive experiments in cognitive robotics*. Paper presented at the Second International Cognitive Robotics Workshop, Berlin, Germany.
- Lewis, M., & Jacobson, J. (2002). Game engines in scientific research. *Communications of the Acm*, 45(1), 27-31.
- Lund, H. H., & Pagliarini, L. (1999). *Robot Soccer with LEGO Mindstorms*.
- Martin, M. C. (2002). *Esc Online: A Venue for Believable Agents*. Carnegie Mellon University.
- Meyer, T. A. (2002). *Development of Computer-Actors within the Interval Script Paradigm*. Unpublished Honours Dissertation, Massey University, Auckland.
- Meyer, T. A. (2003). Synthetic characters on stage. *Playmarket News*, 31, forthcoming.

Meyer, T. A., & Messom, C. H. (2002). *Development of Computer-Actors Using the Interval Script Paradigm*. Paper presented at the 9th Annual New Zealand Engineering & Technology Post Graduate Conference, Auckland, New Zealand.

Open Source Organisation. (2003). *Frequently Asked Questions* [Website]. Retrieved 5 April, 2003, from the World Wide Web: <http://opensource.org/advocacy/faq.php>

Pinhanez, C. (1999). *Representation and Recognition of Action in Interactive Spaces*. Unpublished Doctoral Dissertation, Massachusetts Institute of Technology.

Pinhanez, C. S., & Bobick, A. F. (2002). "It/I": A theater play featuring an autonomous computer character. *Presence-Teleoperators and Virtual Environments*, 11(5), 536-548.

Taylor, P., Black, A. W., & Caley, R. (1998). *Architecture of the Festival Speech Synthesis System*.