USING SYNTHETIC PLAYERS TO GENERATE WORKLOADS FOR NETWORKED MULTIPLAYER GAMES

Asif Raja and Michael Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
E-mail: araja4@csd.uwo.ca, katchab@csd.uwo.ca

KEYWORDS

Networked games, multiplayer games, workload generation, synthetic players, bots.

ABSTRACT

The increase in popularity of online games in recent years has motivated research and development efforts into the creation of new algorithms, architectures, and protocols for these games. Generating suitable workloads to permit live empirical testing and experimentation to verify and validate this work, however, is a difficult process, especially if one is working towards massively multiplayer technologies.

To support these efforts, we have developed a framework for synthetic players that generates appropriate workloads for networked multiplayer games. Based on this framework, we have developed a software infrastructure and a prototype synthetic player for the multiplayer adventure game Crossfire. This paper introduces our framework, discusses our implementation efforts, and presents results from initial experiments using our prototype synthetic player. Results to date have been quite promising.

INTRODUCTION

Video games have been projected to continue to enjoy solid growth over the coming years (PricewaterhouseCoopers LLP 2007). Online video games in particular are continuing to grow in popularity, with reports now estimating that 62% of all video game players play games online (NPD Group 2007). As a result, there is a serious need for research and development efforts into creation of new technologies for networked games that provide improved quality of service to players despite the uncertainties and adversities in network performance and reliability that frequently occur over the public Internet (Carlson et al. 2003).

When creating new algorithms, architectures, and protocols for networked multiplayer and massively multiplayer games, a significant challenge comes in the form of verification and validation of research and development results. Traditionally, this has taken the form of simulation, but the increase in availability of and access to sufficient computing infrastructure and network testbeds in recent years (Craven 2006) now makes live empirical testing and experimentation a realistic and attractive option. A key difficulty in doing this, however, is the generation of a suitable workload for the game in question.

Using human players for workload generation is problematic for several reasons. Coordinating a reasonably large number of human players for specific experimentation can be difficult; this is especially problematic when one would like to generate a workload for a massively multiplayer game potentially requiring thousands upon thousands of players. The use of human players also increases the potential for variability and unanticipated influences on experimentation, making experiments difficult to repeat and replicate, and analyses more complicated and susceptible to error. Because of these issues, the use of synthetic players, driven by some form of artificial intelligence, must be considered very seriously.

This paper introduces a new framework for synthetic players, specifically designed with workload generation for empirical testing and experimentation with networked multiplayer games in mind. This framework supports the creation of independent synthetic players that interface with a networked game in the same fashion as its human players would, to realistically and accurately recreate the workload that human players induce on the game. The framework allows synthetic player behaviour to be scripted or generated dynamically, with generated behaviour being either repeatable or different for every gameplay session. The framework is also flexible and robust, allowing it to be used in constructing synthetic players for a wide variety of games in various genres. Furthermore, it provides the monitoring and control features required to support large-scale rigorous experimentation efforts.

Based on this framework, we have developed software and libraries to facilitate the construction of synthetic players for workload generation for networked multiplayer games. As a proof of concept, we have created a synthetic player for the open-source multiplayer adventure game Crossfire (Wedel et al. 2007), and used this to conduct a variety of experiments with this game. This paper discusses these development efforts, as well as initial experiences with using our synthetic Crossfire player to date.

The remainder of this paper is structured as follows. We begin by discussing related work in this area, providing a brief

overview and analysis of this work. We then describe the framework for synthetic players we have developed, and describe how it can be used in workload generation for networked multiplayer games. We then discuss our implementation efforts and our experiences in using our synthetic Crossfire player to date in experimentation. Finally, we conclude this paper with a summary and a discussion of directions for future work.

RELATED WORK

There has been considerable work done towards the application of various principles and techniques from artificial intelligence to creating realistic synthetic players for video games, as discussed in (Rabin 2002) and elsewhere. Many successes have been achieved for a wide variety of types of games, and on-going work in this area is extremely promising.

Unfortunately, the majority of this work is not suitable for workload generation for multiplayer games. To be suitable, these synthetic players would need to be separate and exist outside of the game as a whole, connecting to the game remotely over the network as a human player would, and acting as if it were a human player in this regard. This is not the case, however, in much of the work in this area. In many cases, in fact, the synthetic players rely on abilities and access to information from the game that are generally not available to human players in order to play the game well (Rouse 2005).

There are, of course, many notable exceptions to this. Interesting work has been done towards the use of synthetic players commonly referred to as bots in a variety of games, from commercial, research, and hobbyist perspectives. This includes bots for games such as Quake (Randar 2007), Quake II (Randar 2007, van Lent et al. 1999), Quake III (Randar 2007, van Waveren 2001), Counter-Strike (Booth 2004), Unreal Tournament (Epic Games 2004), Descent 3 (van Lent et al. 1999), and others.

These bots, unfortunately, tend not to be suitable for workload generation in practice either. While they can connect to a game remotely over a network as a human player would, these synthetic players were designed primarily to provide interesting and challenging opponents to the human players of the game, and not as tools to support empirical testing and experimentation. Consequently, they lack many of the features required to properly support workload generation for this purpose, such as monitoring and control capabilities, as examples.

Furthermore, as one can see from the bots listed above, the majority of bots have been created for first person shooter games. These games have relatively set formulae for gameplay, and tend to not support the rich variety of gameplay and interactions available in other genres like role-playing or adventure games. Consequently, there is a need for more robust and flexible approaches to synthetic players to support a wider variety of games. Since most massively multiplayer online games are role-playing games, and these and games from other genres are among the most popular games currently being played (Nielsen Media Research 2007), this is not something that can be ignored.

As a result, in the end, more work is needed to provide synthetic players to support workload generation for networked multiplayer games.

A FRAMEWORK FOR SYNTHETIC PLAYERS

To support the creation of synthetic players for workload generation for networked games, we have developed a general and flexible framework, as shown in Figure 1. This is based on a sense-think-act process that has been used in artificial intelligence, agent development, and game bots, such as the work in (van Lent et al. 1999). The main elements of this framework are described in the sections that follow.

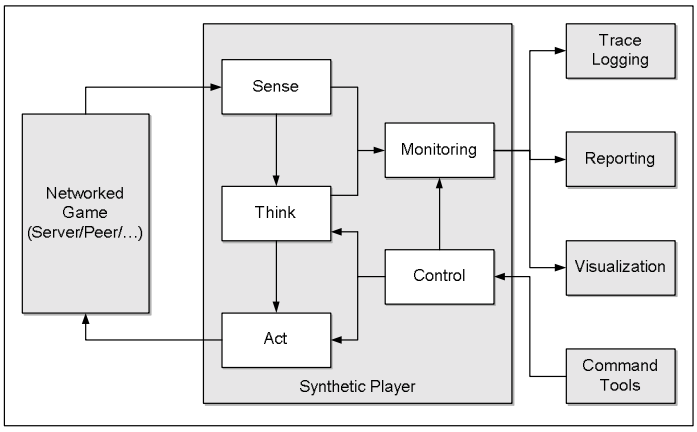

Figure 1: A Framework for Synthetic Players for Workload Generation in Networked Multiplayer Games

Networked Game

The networked game is the game for which workload is being generated by the synthetic player. Ideally, the game should be used without modification from its original form, if at all possible. The networked game can be based on a variety of architectures such as client-server or peer-to-peer; ultimately, it is up to the synthetic player to adapt accordingly to remotely connect and work with the game properly. In the end, the game and other players in the game should be unaware that the synthetic player is not, in fact, a human player after all.

Synthetic Player

The synthetic player is responsible for generating a realistic workload for the networked game in question. It does so by interfacing with the game and playing it in the same manner as a human player would.

The synthetic player consists of two main sets of modules. The first set consists of the sense-think-act logic that enables the synthetic player to properly play the networked game. The second set of modules consists of monitoring and control elements to enable and facilitate experiments using the synthetic player. All of these modules are discussed in the sections below.

Sense

The sense module is responsible for receiving events and messages from the networked game. It uses this information to construct the view of the game and its state that the synthetic player uses in its decision making processes. In theory, this is the same view that a human player would have if it was in the synthetic player's position in the game, although the synthetic player might have access to more information depending on how it interfaces and communicates with the game in question.

Think

The think module is used to encapsulate whatever decision process is used within the synthetic player to guide its actions in playing the networked game and generating its workload. This module relies on the sense module to provide input for the decision process, and has its decisions carried out by passing them onto the act module once they have been formulated.

The decision process used in this module is likely to be some form of rule based system or state machine, but could be virtually any kind of decision making logic. Alternatively, this module could use some kind of scripting system to have the synthetic player generate workload according to a previously scripted sequence of actions. It is also possible to construct a synthetic player with a think module that uses some combination of these approaches as necessary.

Act

The act module is responsible for carrying out the actions decided upon by the think module. It does so by generating appropriate messages and sending them to the networked game according to the formatting and protocol requirements of the game in question. These messages, in the end, are what induce workload on the network game, as the game must receive, process, and respond to these messages appropriately.

Monitoring

The monitoring module is used to observe the synthetic player in playing the networked game and collect and record information about its performance in doing so. Monitoring measures both the workload being generated by the synthetic player and the results of applying the workload to the game. This ensures that the synthetic player is generating the correct workload and provides valuable information that enables qualitative and quantitative evaluation of the experience delivered as a result, as will be discussed further below.

Data being monitored can be collected either from the sense module or the think module. Data elements collected from the sense module are either copies taken from messages received from the networked game, or measurements taken concerning these messages, such as message latency, data volume, and so on. Data collected from the think module, on the other hand, either reports on the synthetic player's decisions or provides the synthetic player's impressions of the game according to its decision logic. The exact data collected, understandably, depends significantly on the game in question.

Data collected through monitoring is directed to modules outside of the synthetic player for a variety of purposes. This includes trace logging, reporting, and visualization, depending on the needs of the experimentation being conducted.

Control

The control module is used to manipulate or tune the behaviour of the synthetic player. This can be done both to influence the sense-think-act logic guiding the decisions made and actions taken by the synthetic player, and to adjust the configuration of the monitoring module, such as what information is collected and how frequently it is collected.

The synthetic player's behaviour in playing the networked game can be directly controlled by manipulating the stream of actions sent to the game by the act module. The synthetic player can also be indirectly controlled by adjusting the decision process used in the think module; for example, by changing internal state, goals, and other elements used in the process. Such control over the synthetic player allows adjustments to generated workloads at run-time, during a test or experiment.

Control is achieved through the use of external command tools that instruct the control module within a synthetic player on what behavioural manipulations are necessary.

Trace Logging

This module is used to create logs of activity from the networked game in real-time, based on data collected by the monitoring module embedded within the synthetic player. The use of the logs depends greatly on the information they contain. For example, a quantitative evaluation of a game can be facilitated by a log consisting of performance measurements over time, such as latency, data volume, anomalies, and so on.

Reporting

The reporting module is used to generate reports of activity from the networked game, typically containing summaries or analyses of data collected by the monitoring module. Again, the use of the reports depends on the information they contain. For example, a report containing summary statistics of various performance measurements, such as those described above, would be quite useful in supporting quantitative evaluations.

Visualization

The visualization module is used to provide a variety of different visual methods of representing data collected by the monitoring module within a synthetic player. These can be as simple as charts or graphs of various data elements, useful for quantitative evaluations, or renderings of the game and its game world, useful for more qualitative evaluations. (In such a case, for example, visualization could provide in essence the same view of the game as would be provided to a human player.)

Command Tools

Command tools are used to exert control over the behaviour of the synthetic player through the use of its embedded control module. Such tools can either be command-line or graphical in nature, depending on requirements for control.

PROTOTYPE IMPLEMENTATION

Based on the framework discussed in the previous section, we have developed software and libraries to ease the construction of synthetic players for workload generation. This includes a layered mapping system as a basis for sense modules to track and record the state of the game world, and a simple and efficient rule based system to be used within think modules to guide decision processes within the synthetic player. Further details of these elements can be found in (Raja 2007).

Using these software elements, we have developed a synthetic player for the multiplayer adventure game Crossfire (Wedel et al. 2007). Crossfire is a fairly mature open-source project that has been developed for several years with an active community of contributors and players. The game has an incredibly rich set of gameplay features allowing its players to interact with each other and the game world in a variety of ways. Crossfire is supported on several computing platforms, including Microsoft Windows, Linux, and many others.

To facilitate our development efforts, we built our synthetic player using the code base for the Linux version of the GTK Crossfire client as a starting point. This allowed us to make use of existing functionality to receive game update messages from a Crossfire server and dispatch actions to the server in our sense and act modules respectively. Since the client is nearly stateless, with almost all game state stored on the server, our sense module made use of our layered mapping system to track the various elements of the game world. (Traditionally, this information would be rendered to the client's graphical interface and not stored at the client.) Human player control was replaced with our own rule based system as a think module.

The original client's graphical interface was kept and preserved to facilitate debugging of the synthetic player, and to act as a rendering system for a visualization module. This allows a human observer to track the game and the quality of the experience being delivered to the synthetic player in real-time. The human player control elements that were replaced by our think module to drive the player were kept for use in a control module to manipulate the synthetic player as necessary during a game. Monitoring, logging, and reporting functions were also added to collect a variety of quantitative performance metrics, including latency and several others.

EXPERIMENTAL RESULTS

To evaluate our approach to workload generation for networked multiplayer games using synthetic players, we conducted a series of tests and experiments using our prototype synthetic player for Crossfire. Initial tests were conducted to verify the synthetic player's ability to play the game and generate a workload, as well as the player's other functionality for monitoring, control, visualization, logging, and so on. After the success of these tests, more thorough performance experimentation was carried out.

Experimentation was conducted in a closed lab environment, making use of 31 workstations connected to the same 100MB switched Ethernet network. Each workstation was powered by a dual core 3.0 GHz Pentium D processor, with 2 GB of physical memory, and Fedora Core 5 Linux as its operating system. One of these workstations was designated to host the Crossfire server for experimentation, while the others hosted synthetic players configured to induce workload on the Crossfire server.

Figure 2 shows performance results from one set of experiments, designed to measure the responsiveness of the Crossfire server as various numbers of synthetic players generated workload. The workload in this case was harsher than one would normally expect, consisting of continuous player motion and conversation with minimal think time between actions. This workload was used as the server proved surprisingly resilient to less intense workloads even with 30 players connected and playing at once. (We are currently investigating how performance holds up with even more synthetic players under lighter workloads, however.)

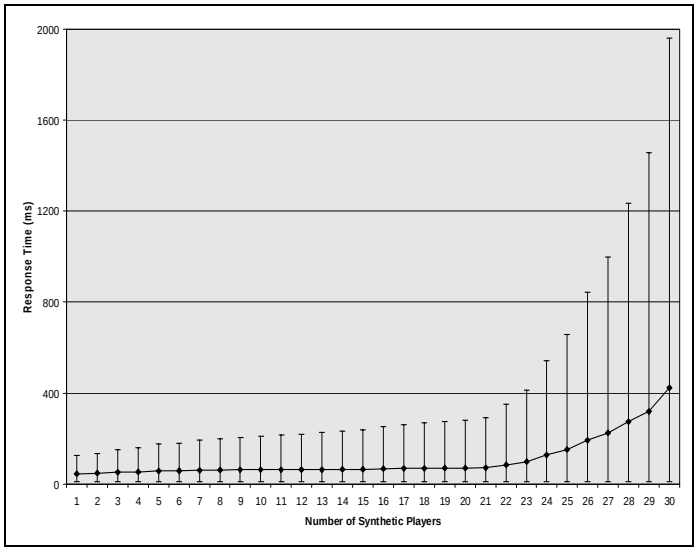

Figure 2: Responsiveness of Crossfire under Increasing Load

The results in Figure 2 were obtained by executing the target number of synthetic players for 10 minutes, during which internal monitoring modules sampled responsiveness every 5 seconds. Responsiveness was calculated as the time required for the Crossfire server to respond to a simple query issued by each synthetic player. Figure 2 presents the mean response time across all samplings from all synthetic players in each 10 minute play interval, along with error bars indicating the minimum and maximum values observed at each load level.

As can be seen from Figure 2, the Crossfire server appeared to do a fairly good job of maintaining a consistent level of performance for up to 21 synthetic players. Past this point, however, response times started to rise dramatically, eventually

reaching an average of more than 400ms at 30 players. This increase in response times is significant and likely more than enough to both impair the ability of players to play the game and affect the ultimate outcome of the game (Armitage 2001). It is also important to note that the variability in response times also increased substantially with more synthetic players in the game, which could further frustrate and annoy human players, had they been playing the game as well.

From a qualitative perspective, we carefully watched the graphical interfaces of the synthetic players as they played the game. As the number of synthetic players in the game increased, we observed a significant increase in the overall delay and jumpiness of the players' displays, especially as the number of players approached 30. This was to be expected considering the increase in response time and response time variation that was measured quantitatively.

In the end, our synthetic players were able to effectively generate workload as instructed. Overall, our synthetic players appear to be very well suited to support testing and experimentation with networked multiplayer games as desired.

CONCLUSIONS AND FUTURE WORK

As networked multiplayer games continue to grow in popularity, it is becoming increasingly important to support research and development efforts in this area to advance the state-of-the-art in both design and technology. Doing so presents many challenges, however, including the generation of suitable workloads to experimentally verify and validate the results of such efforts.

Our current work is aimed at this challenge, providing synthetic players to generate workloads for networked multiplayer games. To this end, we have developed a framework for synthetic players for this purpose, and implemented prototype software based on this framework. In doing so, a synthetic player for the Crossfire multiplayer adventure game was successfully constructed. Experimentation to date using our prototype software has been quite positive and demonstrates great promise for our approach in the future.

There are many possible directions for future work in this area. These include the following:

- Continued experimentation with our synthetic Crossfire player is necessary, particularly with a larger number of players, and under various types of workloads.

- Further development of our synthetic Crossfire player is also possible. This includes adding the option to disable interface visualization elements to permit headless operation, parameterization of personality and behavioural elements to support a wider variety of player types, and extensions to support the incredible variety of gameplay and interactions available within Crossfire.

- Applying our framework to other networked multiplayer games is also important, to help demonstrate its flexibility and suitability in a variety of gameplay situations.

- In particular, applying our framework to a massively multiplayer online game environment is of great interest. This will introduce many challenges in terms of managing and coordinating the large number of synthetic players, as well as in organizing, processing, and analyzing the potentially huge amount of data that can be collected in this kind of scenario.

REFERENCES

Armitage G. 2001. "Sensitivity of Quake3 Players to Network Latency". *Presented at the SIGCOMM Internet Measurement Workshop*. San Francisco, California. (November).

Booth M. 2004. "The Official Counter-Strike Bot". *Presented at the 2004 Game Developers Conference*. San Francisco, California. (March).

Carlson R., Dunigan T., Hobby R., Newman H., Streck J., and Vouk M. 2003. "Strategies & Issues: Measuring End-to-End Internet Performance". *Appeared in Network Magazine*. (April).

Craven D. 2006. Network Testbeds: A Method of Testing Potentially Disruptive Technologies. *MSc Reading Course, Department of Computer Science, The University of Western Ontario*. (February).

Epic Games. 2004. Unreal Tournament 2004. *Published by Atari*. (March).

Nielsen Media Research. 2007. PlayStation 2 Accounted for 42 Percent of Video Game Play in June, Nielsen Reports. *Nielsen News Release* (July).

NPD Group. 2007. Online Gaming 2007: The Virtual Landscape. *NPD Special Report*. (May).

PricewaterhouseCoopers LLP. 2007. Global Entertainment and Media Outlook: 2007-2011. *PWC Report*.

Rabin S. 2002. *AI Game Programming Wisdom*. Charles River Media.

Raja A. 2007. Design and Implementation of an AI Bot for the Adventure Game Crossfire. *MSc Directed Study Report, Department of Computer Science, The University of Western Ontario*. (May).

Randar. 2007. Randar's Bot Page. Available online at: *http://members.cox.net/randar*. (Last accessed August).

Rouse R. 2005. *Game Design Theory and Practice, Second Edition*. Wordware Publishing Inc.

van Waveren J. 2001. The Quake III Arena Bot. *MSc Thesis, Delft University of Technology*. (June).

van Lent M., Laird J., Buckman J., Hartford J., Houchard S., Steinkraus K., Tedrake R. 1999. "Intelligent Agents in Computer Games". *Appeared in the Proceedings of the National Conference on Artificial Intelligence*. Orlando, Florida. (July).

Wedel M. et al. 2007. *Crossfire – The Multiplayer Adventure Game*. (March).