DESIGN AND IMPLEMENTATION OF OPTIMISTIC CONSTRUCTS FOR LATENCY MASKING IN ONLINE VIDEO GAMES

Shayne Burgess and Michael Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
sburges3@uwo.ca, katchab@csd.uwo.ca

KEYWORDS

Latency reduction, latency masking, optimistic execution, software design patterns for games.

ABSTRACT

To achieve interactive experiences comparable to offline games, online video games played over the Internet must be able to deal with performance issues caused by the connection or infrastructure of the underlying network. A particularly difficult issue faced by developers of online games is that of latency. In many cases, the latency encountered forces gameplay to be very frustrating and breaks immersion for the player, providing an unsatisfactory experience overall.

Instead of directly attempting to reduce or eliminate latency from our networks, our approach has been to reduce or eliminate the effects of latency. Our earlier work in this area introduced a framework based on the concept of optimistic execution. In this paper, we discuss the design and implementation of reusable software components based on this framework that are capable of supporting optimistic execution in a wide variety of online video games. This paper also reports on experiences in using these components in the development of a simple trading game to validate their suitability for use in games. These experiences have been quit positive, and demonstrate great promise for future work in this area.

INTRODUCTION

It has recently been projected that video games will see the fastest growth amongst all entertainment markets (PricewaterhouseCoopers LLP 2006). In particular, online video games played over the Internet have been singled out to be among the fastest growing segments within the video game industry (PricewaterhouseCoopers LLP 2006), with 44% of frequent game players playing games online (Entertainment Software Association 2006). As a result, the delivery of high quality experiences to game players will increasingly depend on the ability of game developers to make online games that can cope with the uncertainties and adversities in network performance that frequently occur over the public Internet (Carlson et al. 2003). This, unfortunately, is an exceedingly difficult task.

A particularly challenging problem is that of network latency (Blow 2004). Latency (also commonly referred to as lag) is a time delay that occurs in passing messages through a network. While steps can be taken to reduce latency, it can never be completely eliminated, as a message will always take a non-zero amount of time to propagate through a network. When the network is heavily used to the point of congestion, a frequent occurrence on the Internet (Carlson et al. 2003), latency increases and becomes unpredictable. This can cause disruptions to the flow of an online game, leading to anything from minor annoyance to a totally unplayable experience. Latency has also been experimentally shown to impair player experiences and affect the outcomes in multiplayer games (Armitage 2001), which is highly undesirable.

To address the problem of latency in online video games, many solutions have been proposed. Unfortunately, none of these solutions provide a comprehensive approach that is applicable across all of the wide variety of gameplay elements found in modern video games. While motion and weapons usage can be handled, this is simply not sufficient and rather limiting to the gameplay experiences that can be provided to the player. Furthermore, some of these approaches tend to either induce confusing gameplay or introduce potential inconsistencies or time paradoxes that can break immersion in the game quite easily (Blow 2004). Consequently, a more general, flexible, and robust solution to latency issues is necessary for online video games.

To fill this need, our earlier work introduced New HOPE (Hanna and Katchabaw 2005; Shelley and Katchabaw 2005), a framework for optimistic execution specifically targeted at online video games. The basic premise behind optimistic execution in this case is to allow certain game activities to occur without checking with other parts of the game first, provided that the outcomes of the activities are predictable and recoverable, in case predictions turn out to be incorrect once synchronization occurs. Optimistic execution of such activities occurs in parallel with confirmation of their outcomes, allowing the latency of synchronization to be effectively masked from the player.

Unfortunately, while this earlier work presented a framework for optimism, it did not provide reusable software components that could be used by developers in building their own games that supported optimistic

execution. Instead, developers had to follow the framework and build everything themselves, as it was not originally thought that general purpose optimistic constructs were feasible (Hanna and Katchabaw 2005).

Our current work remedies this situation through the introduction of reusable software components to provide the necessary supports for optimistic execution for latency masking. This was made possible through the design and refinement of new software patterns for optimism, based on our earlier work from (Hanna and Katchabaw 2005; Shelley and Katchabaw 2005), and their implementation as a collection of .NET managed objects. To validate the effectiveness of these new patterns and software components, and to demonstrate their usefulness, we have developed a simple trading game, Space Traders, as a proof of concept.

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 each approach and technique. We then describe the pattern-based design of our new optimistic constructs and their implementation as reusable software components. We then present our proof of concept trading game, and discuss our experiences in using our software components in its construction. Finally, we conclude this paper with a summary and a discussion of directions for future work.

RELATED WORK

Our approach to optimistic execution is an evolution of the first HOPE (Hopefully Optimistic Programming Environment) project (Cowan 1995), originally designed for non real-time applications. HOPE made exclusive use of rollback to recover from situations in which incorrect optimistic predictions were made. Unfortunately, the exclusive use of rollback makes HOPE not suitable for networked multiplayer games. A total rollback of activity would effectively undo player actions and reactions, in essence moving the game backwards in time, which is highly undesirable in general. Game progression, simply put, must always go forward in time.

Dead reckoning, discussed in (Aronson 1997), is a classic method that can be used for predicting and extrapolating the behaviour of entities in a game world based on algorithms and models of movement and physics in the game. The work in (Bernier 2001) discusses similar prediction techniques, specifically applied to the game Half-Life. With accurate prediction, such methods can be quite effective. When predictions are found to deviate from reality, corrections are made that may cause a snap in player position, as the old, incorrect position is updated with the newly corrected position. This can cause serious and noticeable problems, particularly in action-oriented games (Pantel and Wolf 2002b). Smoothing algorithms can be used to minimize this snapping effect, at the cost of delayed synchronization of game states.

There have been many extensions to dead reckoning and client-side prediction techniques. The work in (Aggarwal et al. 2004) and (Mauve 2000) is aimed at improving accuracy in predictions, but does so at the cost of requiring global synchronization or increased message traffic and complexity. Context based reckoning, introduced in (Schirra 2001), is a method in which natural language is used to convey game activity instead of numeric and geometric data traditionally used. This requires special techniques to both identify and encode game events, and other techniques to decode them for use. Context based reckoning shows promise, but is complex and potentially unreliable, particularly if errors occur in the encoding or decoding phases.

Presentation delay (Pantel and Wolf 2002a) is a technique in which processing and presentation of game events in local and remote entities are synchronized. This requires that local events are delayed. While this can remove inconsistency problems, a serious issue introduced by latency in games, this comes at the cost of additional delays; experimental results presented in (Pantel and Wolf 2002a) and further examined in (Pantel and Wolf 2002b) indicate that this approach can produce unacceptable results in time sensitive action-oriented games.

Local perception filters were used in (Smed et al. 2004) as a technique for implementing “bullet time” in multiplayer games. These filters can also be used in a game for masking latency by allowing temporal distortions in the rendered view of the game. In essence, different parts of the game world are allowed to be rendered at different times, depending on the proximity and possibility of interaction between the various entities in the world. While showing improvements in certain gameplay scenarios, local perception filters require that exact communication delays are known, and exhibit disruptions in the game when sudden changes to the game world occur (such as when one player in a multiplayer game exits the world).

Server-side techniques for masking latency can be found in (Fraser 2000) and (Bernier 2001) for Unreal Tournament and Half-Life respectively. This approach to latency compensation can be thought of as a step back in time. Suppose a player invokes some action and this event is forwarded to a game server for processing. The server computes latency, and deduces the time at which this action was invoked. The server then moves the state of the game world back to this time to determine the effects of the action, applies the action, and moves the state back to its current condition. While this technique can be effective, it does introduce other paradoxes into the game world that can be difficult to handle and produce their own problems, as discussed in (Fraser 2000) in detail.

While several potential solutions to the problem of latency in networked multiplayer games have been proposed, each has its own drawbacks and limitations. In particular, these approaches tend to focus on movement and shooting aspects of first person shooters, and other similar games. Some solve certain latency-related problems, but do so at the risk of introducing new problems,

inconsistencies or paradoxes at the same time. Our approach differs in that it is a more general and flexible solution, capable of supporting more varied gameplay. In following our approach, developers are forced into dealing with the issues introduced while masking latency, and are given appropriate tools to be able to address and resolve these issues in a manner acceptable to the players of the game. This is discussed further later in this paper.

OPTIMISTIC CONSTRUCTS FOR ONLINE VIDEO GAMES

In this section, we discuss the design and implementation of our optimistic constructs to mask latency in online video games. These constructs are derived from our earlier work in (Hanna and Katchabaw 2005; Shelley and Katchabaw 2005), which has been refined to provide components that are reusable in wide variety of games and gameplay mechanics, and that can be easily implemented as a portable class library to assist developers in creating online games supporting optimistic execution.

Pattern-Based Design of Optimistic Constructs

Our earlier work in (Shelley and Katchabaw 2005) introduced an overall framework for optimistic execution in games, loosely based on the concept of software patterns (Gamma et al. 1995), but lacking much of the rigor and detail traditionally used in such patterns. While this was consistent with other software patterns developed for games (Björk and Holopainen 2005), it only provided the main concepts behind optimistic execution. This allowed developers to create online games that made use of optimistic execution, but only in an ad-hoc fashion, treating each game as a separate application of the framework pattern, effectively starting from scratch each time.

To rectify this situation, a thorough and detailed set of software patterns were developed to provide a set of optimistic constructs for online video games. In doing so, we were able to provide a set of reusable software components for optimistic execution in online games that are capable of effectively masking latency encountered during execution.

Figure 1 depicts the main elements of our new design. This includes the following optimistic constructs: actions, recovery modules, padding modules, synchronization modules, and decision modules. These key elements are discussed briefly in the remainder of this section. For full details of the software patterns in the standard format traditionally used by software patterns (Gamma et al. 1995), the reader is urged to consult (Burgess 2006).

Actions

For the purposes of our work, a video game is driven by a series of actions. These can be generated by player characters, for example moving, shooting, and interacting with objects or other characters; by non player characters, in exhibiting similar behaviours to player characters; or by other elements in the game world, handling non-character driven activities. The results of actions change the state of the game world and its inhabitants and consequently must be propagated to all players of the game as necessary to ensure that everyone has a consistent view of the game. Otherwise, the inconsistencies in the game can lead to player frustration, a loss of player immersion, and an overall negative gameplay experience. Actions can be handled and processed within a game in one of two ways, optimistically or cautiously.

If the results of an action are reasonably predictable, and can be recovered from if necessary, optimistic execution is the best approach. In this case, the predicted results of the action are assumed to be true, and execution proceeds based on this assumption while verification of the results proceeds in the background. Since we are concerned with online games, this verification process will likely entail network communication and remote computation of some kind to yield the actual results of the action. If the assumption is later found to be correct, execution can continue, and the latency of verifying the results of the action is effectively hidden, since the game did not have to pause and wait during this process. However, if the assumption is found to be incorrect, the execution of the game since the assumption was also incorrect, and the game will need to execute a recovery to bring all parts of the game back into an acceptable and consistent state. If recoveries are needed only rarely and do not disrupt the flow or immersion of the game, this approach can be quite effective in masking latency.

If the results of an action cannot be reasonably predicted, or cannot be recovered from easily, it is better to process the action in a cautious fashion, instead of proceeding optimistically. This requires that the results of the action are verified before the game proceeds with execution, which makes latency in the required network communication and remote computation potentially visible to the player. However, this may be necessary to prevent excessive recoveries or to avoid situations from which recoveries are not possible, as these conditions could very well be worse to the player than a more cautious execution.

The Recovery Module

Recoveries are used to bring a game back into an acceptable state following the denial of an optimistic assumption. If a recovery is not carried out, the various elements of the game will not be in agreement over the outcome of the action that was processed optimistically, and the resulting inconsistencies could have a very serious impact on the game as a whole.

Since multiple recoveries from a denied optimistic assumption may be possible, a recovery selection procedure must be followed to determine the best recovery to handle the current situation. The selection of recovery method can depend upon many factors. These include the original action executed, the optimistic execution that was carried out afterwards, as well as a variety of game and action specific factors.

After the execution of this recovery, the game is allowed to proceed from this corrected state.

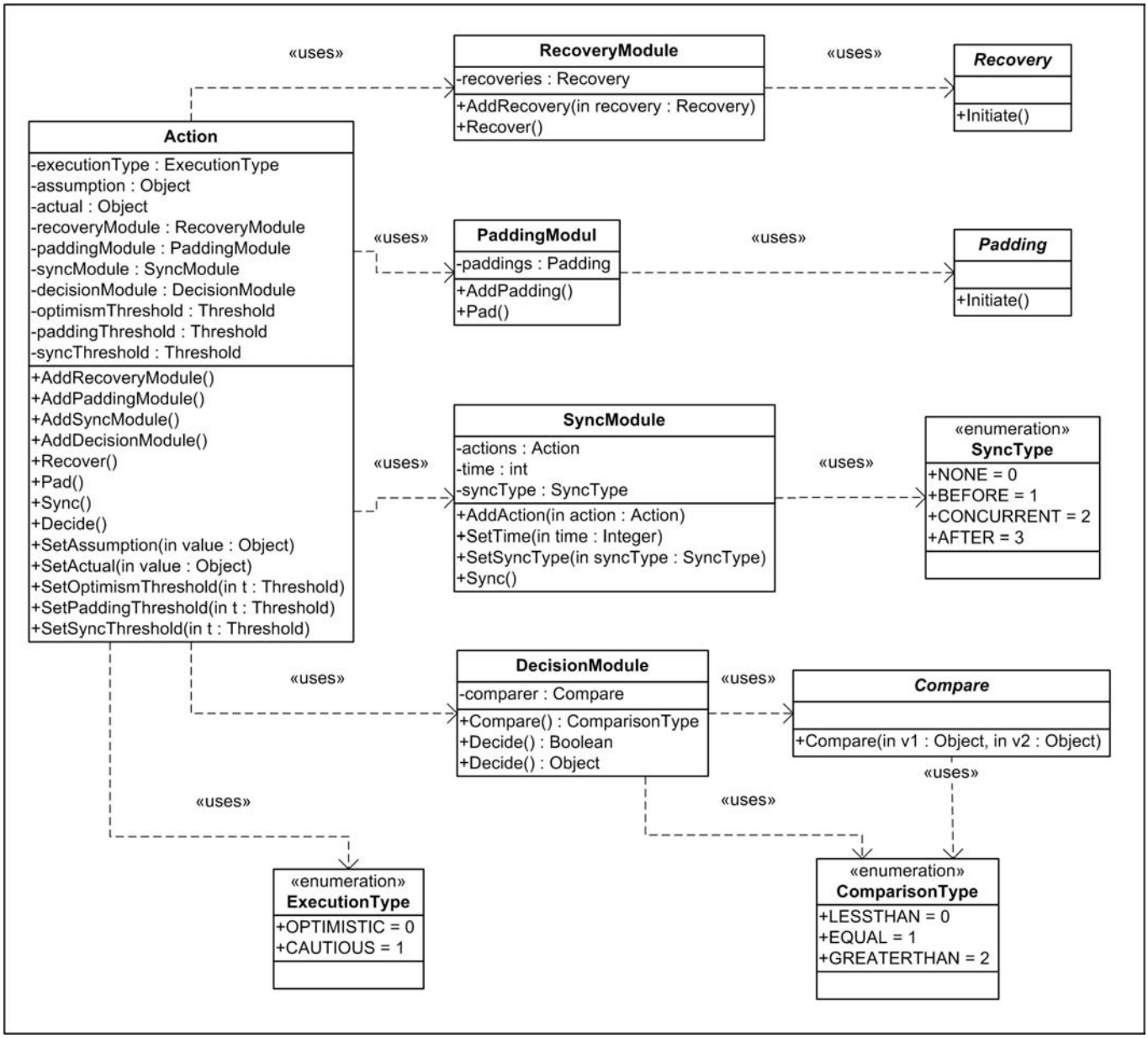

Figure 1: Optimistic Constructs for Online Video Games

The Padding Module

Padding is used to add some form of distraction element to the game to either mask a cautious execution or reduce the amount of optimistic execution that occurs. Padding can be as simple as an animation played to consume a small amount of game time, or can be considerably more complex, depending on the game in question. Padding can be used in a wide variety of situations, but is particularly useful when the recoverability or predictability of an action is below a threshold of comfort and still somewhat questionable as a result.

Before employing padding, a decision process must be followed to determine whether or not padding is appropriate in the current situation within the game. After reaching a decision that padding is necessary, it must also be determined which methods of padding are appropriate in this situation so that one can be selected accordingly. (Multiple methods of padding should be provided to handle different situations, and to allow for variety in the handling of the same situation multiple times to avoid unwanted and noticeable repetition.) The padding is then executed, and optimistic or cautious processing continues upon the completion of the padding. Either way, the distraction element in the padding effectively masks the latency of result computation and communication that is occurring in parallel.

It is important to note that padding may consume either a part or all of the time that could have been spent executing optimistically or pausing cautiously, depending on the situation and the padding involved. (It is not a good idea for padding to take longer than this, however, as this could slow the pace of the game unnecessarily, be disruptive, and lead to player frustration.) Furthermore, by employing padding, recovery from optimistic execution is lessened if

the original assumption was incorrect, because the amount of execution was itself lessened.

The Synchronization Module

The synchronization module is used to provide synchronization primitives for optimism. Synchronization constraints can be added to an action to force execution to wait before or after the action for the results of another action or set of actions to be confirmed. This can be used to prevent further optimistic execution from proceeding if that execution would be difficult to recover from. Time delays can also be used in this process if necessary. It is important to note that recovery would still be necessary upon denial for any optimistic execution up until this point, however.

For example, suppose the player picks up an object and then attempts to use it. If the action of picking up the object was executed optimistically, the act of using the object likely needs to be synchronized to prevent the use of an object that was not actually picked up, in case the optimistic assumption was later denied. Otherwise, this could introduce problematic inconsistencies and paradoxes into the game.

The Decision Module

This module is used to facilitate various decisions governing the optimistic execution of a particular action. This includes decisions on whether to execute optimistically or cautiously, whether to employ padding or not, and whether to force synchronization or not. (Decisions as to which recovery to use when recovery is necessary, or which padding to use when padding is necessary, are up to those modules to make.)

This decision making processes will weigh several game and action specific factors against one another and derive measures of recoverability and predictability; these measures are then compared against thresholds to determine how execution should proceed. Players should be given input over the setting of these thresholds to tune gameplay to their own preferences and tolerances, although the game should have some input as well, according to observed latency in the network. By allowing a choice between optimistic and cautious execution at run-time, finer control over optimism can be achieved, and a better play experience can be provided to the player. (As warranted, static decisions can be embedded for performance reasons, to avoid overhead in the decision processes when optimism clearly should or should not be used.)

Implementation of Optimistic Constructs

The optimistic constructs described above were implemented so that they could be reused in a wide variety of games without having to re-implement the constructs each time. The implementation was programmed in C# using .NET managed objects. While this means that these optimistic constructs can be used in any .NET-aware game regardless of the language used in creating the game, this does hamper their use in games that are not .NET-aware, without the use of some kind of software wrapper. Given the increase in use of .NET among developers, this is not likely to be an issue.

Most of the optimistic constructs discussed in the previous section can be used and reused with no modifications required, although since the implementation is object-oriented, it is possible to specialize these constructs if needed for particular games. Only three of the constructs depicted in Figure 1 must contain game-specific operations that cannot be carried out in a simple and generic fashion. To handle these cases, our implementation relies on a number of abstract classes that have to be implemented before the constructs can be used. (This is a common feature of many design patterns, and allows them to be both reusable and flexible (Gamma et al. 1995).) These abstract classes define the Recovery, Padding, and Compare constructs.

Recoveries and padding, by their very nature, are game-specific and must be created by the game's developers. To do so, developers derive new classes containing implementation details specific to their games from the abstract classes provided by our class library. When a recovery or padding is required, the appropriate initiation method is invoked by the recovery or padding module respectively, causing the game-specific code to be executed. This game specific code could then do whatever is necessary to either carry out a recovery or perform a padding operation within the game. In this way, the generic optimistic constructs provided by our class library can still support optimistic handling of actions in a game-specific fashion.

Compare constructs are used to evaluate and compare various Threshold objects used by the Decision Module in making its decisions; these constructs can also be game-specific. Consequently, developers will need to provide appropriate comparison classes for game-specific situations, again derived from the abstract classes provided by our class library. Our class library also provides concrete comparison constructs for common types used in comparisons, to ease development.

Once the required recovery, padding, and comparison elements are implemented and provided, they seamlessly integrate and work with the other optimistic constructs in our class library.

PROOF OF CONCEPT: SPACE TRADERS

As proof of concept, the Space Traders game was developed using the optimistic constructs described in the previous section.

Overview of Space Traders

Space Traders is a simple trading game in which the players travel the universe, visiting planets to buy and sell resources to accumulate as much wealth as possible in the process. Each planet that the players travel to in the game

has set prices for the various resources and set quantities of each that the players can purchase. The prices of these resources change as they are purchased by the players visiting the planet. Traveling from planet to planet also costs fuel which players must purchase as necessary. This game was developed using Microsoft's Visual Studio .NET, and programmed in C#.

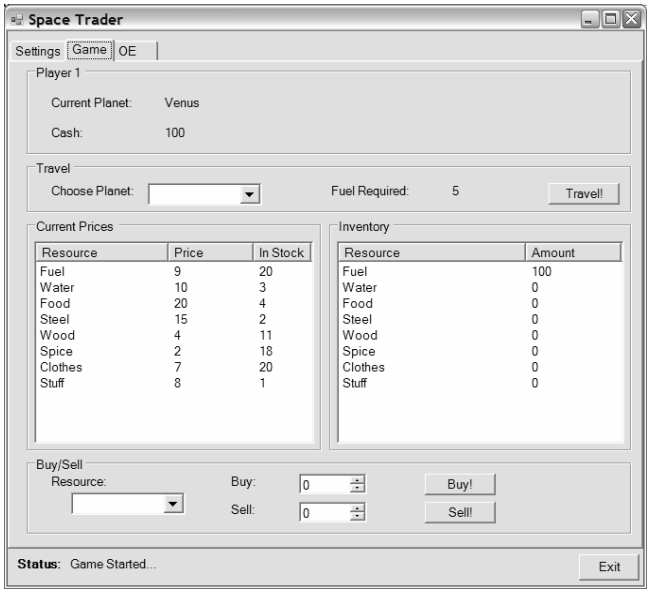

Figure 2: Screenshot of Space Traders Client

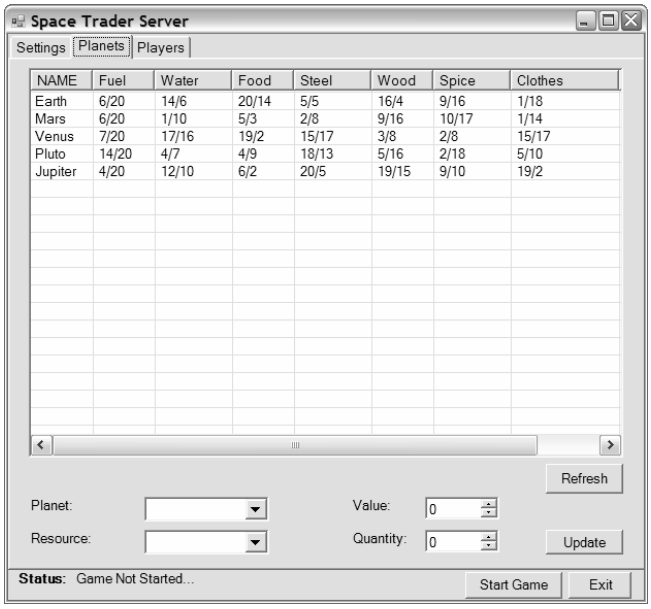

Figure 3: Screenshot of Space Traders Server

Space Traders uses a client-server architecture; screenshots from both the client and server are shown in Figure 2 and Figure 3 respectively. The TCP transport protocol is used for communication between the client and the server. The server is responsible for maintaining the game's state and updating it according to player input data received from the clients. This includes calculating new price and resource availability, depending on the buying and selling patterns of the players in the game. (In essence, the more abundant a resource is, the lower its price will be, and the scarcer a resource is, the higher its price will be.) This updated game state is then sent back to the clients. At the clients, updated game states related to the last player input are rendered to the display as they are received.

Optimistic Execution in Space Traders

In Space Traders, each player has three main actions they can choose from: traveling from one planet to another, buying resources at their current location, or selling resources at their current location. As the player carries out these actions, they obtain feedback on their outcomes. (As mentioned earlier, clients also periodically receive updates on resource prices and availability when changes occur at their current location.)

The travel action is always carried out in a cautious fashion as the client requires a listing of resource prices and availability at its new location before proceeding. Because of the nature of this information, there are simply no reasonable optimistic assumptions that can be made for this action. Buy and sell actions, however, can be made optimistically, under the assumption that the resource price and availability information that the client has is still current and up-to-date. This may or may not be a good assumption for the client to make, as it turns out.

The fluctuations in resource prices and availability represent a potential source of inconsistencies in the game. When making a transaction to buy or sell a particular resource, it is in fact quite possible for both its price and availability to change between the last update in information received by the player's client and the initiation of the transaction, meaning that the player is conducting business with an out-of-date view of the game world. This is particularly the case when several players are visiting the same world, conducting transactions at the same time, as the handling of these transactions will cause such changes to occur. If any buy or sell actions are carried out with incorrect resource prices or availability, the optimistic execution of these actions would be incorrect as well.

A decision module is used to make an initial decision about using optimism. If the results of a transaction are predictable, because there are few other traders on the planet to influence the price and availability of resources, then transactions will proceed optimistically. Otherwise, they will be carried out cautiously. (With too many traders on the same planet, the possibility for resource price and availability changes becomes unacceptably high and too many incorrect optimistic assumptions will require recoveries of some kind to correct.)

If an assumption about the results of a transaction is incorrect, a recovery process is initiated to correct the situation in a fashion consistent with the rest of the game. For example, suppose a poorly-timed change in price

caused a player to overpay for a resource in a transaction. Suppose that the last update from the server to Player 1's client prices the resource water at $10 a unit, with 3 units available for purchase, as shown in Figure 2. Now suppose that while Player 1 is making a purchase decision, Player 2 sells an additional 6 units of water on the same planet, causing the price of water to drop to $5 a unit. If Player 1 decides to purchase what they believe is all the water on the planet before receiving an updated resource list, Player 1's client will mistakenly approve a purchase of 3 units of water at $10 a unit, instead of the $5 a unit it actually cost. Since the buy action executed by Player 1 is optimistic, the player's client will process the transaction and believe the player has less money than they actually do because it is unaware of the inconsistency between its resource list and the actual list for the planet stored at the server. When the optimistic execution of this buy action is found to be incorrect, a recovery is taken to give the player their money back and correct the mistaken optimistic assumption. This can be done through a simple message, such as the one depicted in Figure 4.

Figure 4: Screenshot of a Recovery Message in Space Traders

By having several possible messages to account for incorrect resource price and availability assumptions during buy and sell actions, several different recoveries are possible. Naturally, seeing these messages pop up too frequently for recovery purposes will begin to adversely affect the player's overall experience in the game. This is why decisions to proceed optimistically should be made carefully depending on the likelihood of success of the actions in question.

Padding and synchronization elements are also used where appropriate within the game. For example, several passing messages were developed as options for display when a buy or sell action had to be processed cautiously instead of optimistically, due to the number of other players on the same planet at the same time. By the time the user reads the message and clicks the "OK" button to dismiss the message, it is likely that sufficient time has lapsed to cover the cautious execution of the action with the server, and the latency of communication is still effectively hidden.

All optimistic execution described above is accomplished using the reusable optimistic constructs described in the previous section. No programming was required to support this optimistic execution, except for providing appropriate recovery, padding, and comparison mechanisms, and to link the optimistic constructs into the rest of the game's code.

Experiences with using our optimistic constructs in developing Space Traders were quite positive. The constructs provided an excellent framework for building optimism into the game, greatly facilitating and easing the development process. Once complete, the optimistic execution within the game worked as expected, masking the latency of communication between clients and the server. Initial experimentation has indicated that latencies up to 200ms can be hidden through the above use of optimistic constructs, with little or no perceptible impact on gameplay. More thorough and rigorous experimentation with a broader player base is currently under way to further investigate the latency masking capabilities of our optimistic constructs.

Based on these results, it is expected that other developers can use these optimistic constructs to add optimistic execution to online video games successfully and easily. Consequently, these constructs could prove quite useful to reducing the effects of latency in games.

CONCLUSIONS AND FUTURE WORK

Latency is a challenging problem to the development and success of online video games. Our current work is aimed at reducing or eliminating the effects of latency to produce more enjoyable gaming experiences for players. Through the optimistic constructs designed and implemented in this work, an important and powerful tool is given to game developers to integrate optimistic execution into their own games. Our own experiences in using these constructs in the development of a simple trading game, Space Traders, have shown their usefulness, and demonstrate great promise for the future.

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

- Further experimentation with our optimistic constructs is clearly necessary. We need to fully investigate the latency reduction benefits of optimism in a variety of online games under a variety of network conditions, and learn how to further tune the factors influencing optimism decisions to improve performance.

- Further study is also required into the use of both nested optimistic assumptions and feedback to tune the decision processes used within the optimistic constructs, as discussed in (Shelley and Katchabaw 2005). Neither of these elements was used in the development of the initial prototype of Space Traders, and so implementation and experimentation efforts are currently under way.

- Many of approaches to latency compensation discussed earlier, including dead reckoning and so on, have predictive elements that, in the end, make them similar to the constructs used in optimistic execution that have been discussed in this paper. Consequently, in the future, we plan to use the optimistic constructs introduced in this paper to re-implement these approaches within this framework. Not only will this provide further validation of this work, but it will also demonstrate its power and flexibility.

REFERENCES

Aggarwal S., H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan. 2004. "Accuracy in Dead-Reckoning Based Distributed Multi-Player Games". *Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04: Network and System Support for Games*. Portland, Oregon. (August).

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

Aronson J. 1997. "Dead Reckoning: Latency Hiding for Networked Games." *Appeared in Gamasutra. Available online from Gamasutra's website at http://www.gamasutra.com/features/19970919/aronson_01.htm.* (September).

Bernier Y. 2001. "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization." *Presented at the 2001 Game Developers Conference*. San Francisco, California. (March).

Björk S. and J. Holopainen. 2005. *Patterns in Game Design*. Charles River Media.

Blow J. 2004. "Miscellaneous Rants". *Appeared in Game Developer Magazine*. (May).

Burgess S. 2006. Patterns for Optimism for Reducing the Effects of Latency in Networked Multiplayer Games. *Undergraduate Thesis, Department of Computer Science, The University of Western Ontario.* (March).

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

Cowan C. 1995. "A Programming Model for Optimism". PhD Thesis. Department of Computer Science, The University of Western Ontario. (February).

Entertainment Software Association. 2006. Essential *Facts about the Computer and Video Game Industry*. Entertainment Software Association Research Report. (April).

Fraser J. 2000. "Zeroping Frequently Asked Questions". *Accessible online at: http://zeroping.home.att.net.* (April).

Gamma E., R. Helm, R. Johnson, and J Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hanna R. and M. Katchabaw. 2005. "Bringing New HOPE to Networked Games: Using Optimistic Execution to Improve Quality of Service". *In the Proceedings of the DiGRA 2005 Conference.* Vancouver, Canada. (June).

Pantel L. and L. Wolf. 2002a. "On the Impact of Delay on Real-Time Multiplayer Games". *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. Miami, Florida. (May).

Pantel L. and L. Wolf. 2002b. "On the Suitability of Dead Reckoning Schemes for Games". *Proceedings of the First Workshop on Network and System Support for Games*. Bruanschweig, Germany. (April).

Mauve M. 2000. "How to Keep a Dead Man from Shooting". *Lecture Notes in Computer Science; Vol. 1905. Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*. Enschede, Netherlands. (October).

PricewaterhouseCoopers LLP. 2006. Global Entertainment and Media Outlook: 2006-2010. *PWC Report*.

Schirra J. 2001 "Content-Based Reckoning for Internet Games". *Proceedings of the Second International Conference on Intelligent Games and Simulation (GAME-ON 2001)*. London, England. (November).

Shelley G. and M. Katchabaw. 2005. "Patterns of Optimism for Reducing the Effects of Latency in Networked Multiplayer Games". *In the Proceedings of the FuturePlay 2005 Conference*. East Lansing, Michigan. (October).

Smed J., H. Niinisalo, and H. Hakonen. 2004. "Realizing Bullet Time Effect in Multiplayer Games with Local Perception Filters". *Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04: Network and System Support for Games*. Portland, Oregon, (August).