Patterns of Optimism for Reducing the Effects of Latency in Networked Multiplayer Games

Gabriel Shelley and Michael Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
gshelley@uwo.ca, katchab@csd.uwo.ca

Abstract

The video game industry has evolved in such a way that many users not only want, but also expect some form of multiplayer experience in games. More so, users anticipate the same quality of service online as they do offline, regardless of the limitations in the connection or infrastructure of the underlying network. This expectation is especially problematic in highly time sensitive multi-player games such as first person shooters and sports games. In many cases, the latency encountered forces gameplay to be very frustrating and breaks immersion for the player. While there have been solutions proposed to help mitigate this problem, they tend to focus on some particular game genre or gameplay element.

To address this issue, this paper presents a new approach to reducing the effects of latency in networked multiplayer games that relies upon techniques in optimistic programming. In particular, this paper introduces software design patterns for building optimistic constructs into networked games, and reports on experiences in using these patterns in the development of a simple football game to validate their use in networked games.

Keywords

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

1. Introduction

With online games continuing to be the fastest growing market segment in the video game industry [16], providing players a satisfactory gameplay experience will increasingly depend on the underlying network and its overall performance. To cope with this situation, game developers require comprehensive and effective methods to reduce the impact of adverse network conditions on their games that can occur far too often [8].

Latency (also commonly referred to as lag or end-to-end delay) is an especially challenging problem [7], 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 [3]. In some respects, unfortunately, there is little that can be done, particularly for games played over wide area networks such as the Internet. Ultimately, the speed of light is not amenable to change.

There have been several proposed solutions introduced to address this problem. Unfortunately, most of these solutions tend to be very narrow and very ad hoc, applying only to one aspect of a single genre of games. Furthermore, some of these approaches tend to either induce confusing gameplay or introduce potential inconsistencies that can break immersion in the game quite easily [7]. With more varied gameplay from a wider variety of genres moving online, a more general, flexible, and robust solution is necessary.

To fill this need, our earlier work introduced New HOPE [12], a framework for optimistic execution specifically targeted at networked multiplayer 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 hidden from the player.

Our current work builds upon the principles of New HOPE introduced in [12], providing software design patterns for optimism in networked multiplayer games. By developing design patterns, we can identify the structural elements required for optimism independent of genre and gameplay, and provide practical implementation guidelines for the construction of networked games that use optimistic execution to reduce the effects of latency. Based on these patterns of optimism, we have developed a simple football game, Football Invaders, as a proof of concept to demonstrate the effectiveness and usefulness of our work.

The remainder of this paper is structured as follows. Section 2 discusses related work in this area, providing a brief overview and analysis of each approach and technique. Section 3 introduces and describes the patterns of optimism developed in this work and provides guidelines for their use in developing networked multiplayer games. Section 4 presents our proof of concept football game, and discusses our experiences in using our newly developed design patterns in its construction. Finally, Section 5 concludes this paper with a summary and a discussion of directions for future work.

2. Related Work

New HOPE is an evolution of the first HOPE (Hopefully Optimistic Programming Environment) project [9], originally designed for non real-time applications. HOPE made exclusive use of rollback to recover from situations in which incorrect optimistic predictions were made. While this was well suited for its target applications, primarily banking systems and other transaction-oriented systems, it also made HOPE not suitable for networked multiplayer games. A total rollback of activity would be tantamount to undoing player actions and reactions, effectively 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 [4], is a 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 [5] discusses similar prediction techniques, specifically applied to the game Half-Life. When predictions work well, 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 problems, particularly in action-oriented games [14]. 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 [1] and [15] 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 [17], 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 [13] 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 [13] and further examined in [14] indicate that this approach can produce unacceptable results in time sensitive action-oriented games.

Local perception filters were used in [19] 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 [10] and [5] 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 [10] 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. New HOPE differs in that it is a more general and flexible solution, capable of supporting more varied gameplay. This is discussed further in the next section.

3. Patterns in Optimism for Latency Reduction

In this section, we introduce software design patterns for optimism to reduce the effects of latency in networked multiplayer games. Before doing so, we first describe software design patterns in general and motivate their use in game development.

3.1. Software Design Patterns

The concept of design patterns was originally introduced in [2], and defined as follows: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." While these patterns were architectural patterns in buildings and towns, many in software design have applied this same concept in formulating software design patterns. As discussed in [11], software design patterns provide the structure for a design solution, including the elements that make up the design and their relationships, responsibilities, and collaborations. Concrete design details or implementation specifics are not provided, as a pattern is intended to act as a blueprint or template that can be used in a variety of situations.

The use of patterns in game development is not new. For example, [6] provides a collection of over two hundred game design patterns for various aspects of games and gameplay. Game design patterns can provide game developers with templated solutions to a variety of problems, independent of genre, rules, objectives, story, characters, platform, and so on. Consequently, if used properly, they can serve a very useful purpose in the development of a game.

3.2. Software Design Patterns for Optimism

In [12], we introduced New HOPE as a framework for optimism to reduce the effects of latency in networked games. To facilitate the development of optimistic games, we have developed several software design patterns based on elements of this framework. The patterns themselves are too lengthy and detailed for full inclusion here. In this paper, we present an overall pattern for optimism and discuss this pattern and its sub-patterns at length, as the overall pattern provides sufficient information for most purposes. For complete details on all of the sub-patterns, in a standard pattern form, the reader is urged to consult [18].

The overall pattern for optimism in networked games is given in Figure 1. Instead of a standard object diagram in OMT or UML, we instead provide more of a flow diagram to illustrate the important elements of the patterns, their relationships, and the flow of control required to produce optimistic behaviour. The various sub-patterns and elements of this pattern are discussed in further detail in the remainder of this section. A concrete example of many of these concepts is provided in Section 4 in the discussion of the proof of concept football game, Football Invaders.

3.2.1. The Optimism Decision Sub-Pattern

The optimism decision sub-pattern is used to make a choice as to whether proceed with execution in the more traditional pessimistic fashion, or in an optimistic fashion. If pessimistic execution is selected, execution will have to block to wait for the results of the action executed to be computed before proceeding. If optimistic execution is selected instead, an assumption will be made about the outcome of the action, and execution will proceed based on this assumption. At the same time, verification of the assumption will be executed in parallel. If the assumption was correct, the latency in verifying the outcome of the action is effectively hidden; if incorrect, a recovery method will need to be executed.

In essence, this decision amounts to determining if the action to be invoked is sufficiently recoverable and predictable to permit optimistic execution. If the action is not easily recoverable, the consequences could be disastrous if the assumption made was incorrect. If the outcome of the action is not very predictable, then a recovery is more likely to be necessary. Since recoveries can be more costly and more jarring than pessimistic execution would have been, this situation should be avoided when possible.

This decision making process 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 pessimistic and optimistic 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.)

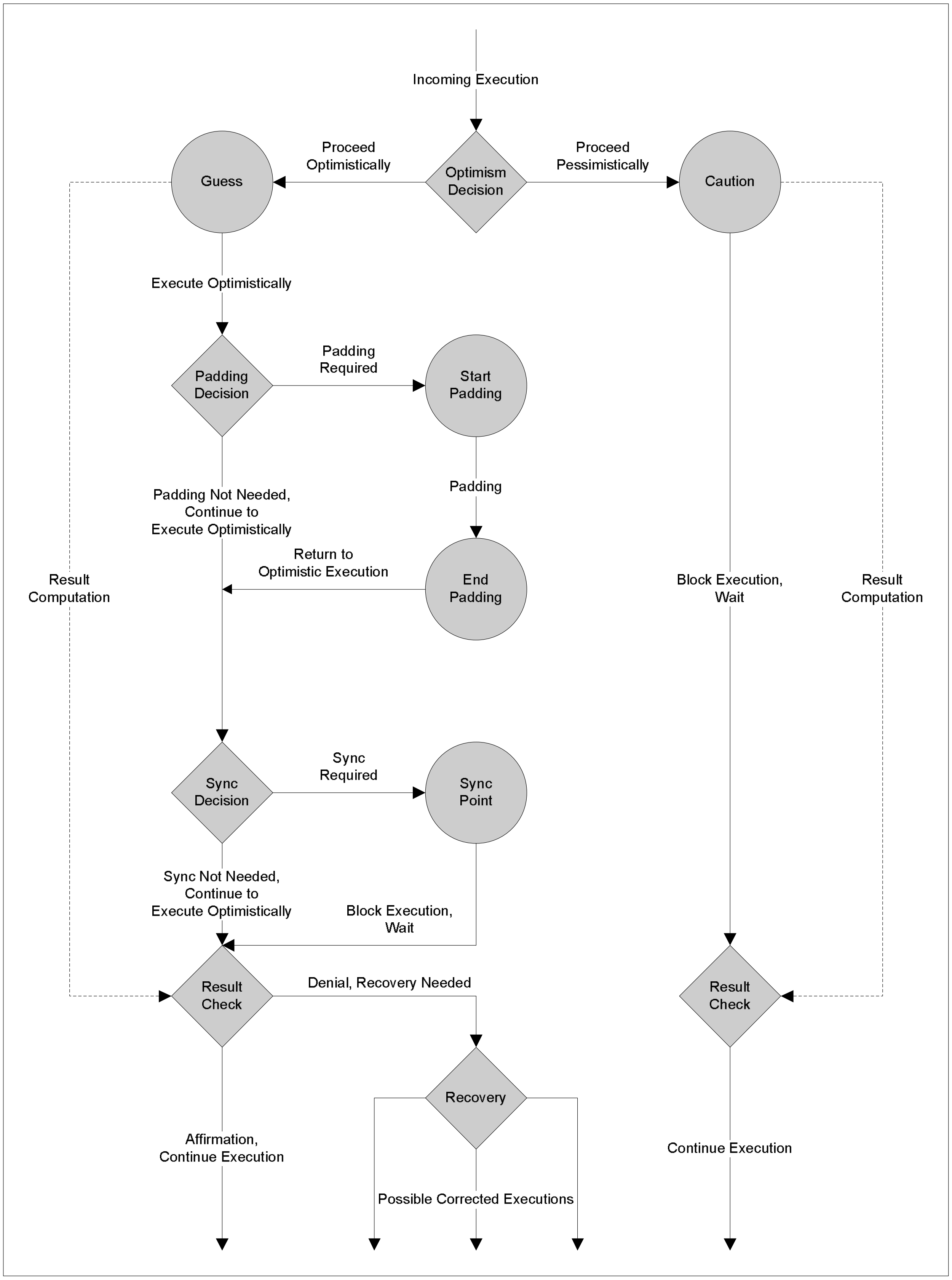

Figure 1. Overall Pattern for Optimism in Networked Multiplayer Games

3.2.2. The Caution Sub-Pattern

The caution sub-pattern involves the pessimistic execution of an action within a game to ensure that the outcome of the action must be known before proceeding. Since we are focusing on networked games, this will require communication between a local entity (typically a client) and a remote entity (typically a server) to determine the outcome of the action executed. While waiting for the remote computation to complete and return its result, local execution must block and wait. This can result in a noticeable break in gameplay and produce an unsatisfactory experience for the player, particularly if the latency of communication is very large.

While this method of execution can produce unsatisfactory experiences, the use of the caution sub-pattern is at times necessary and unavoidable, particularly for actions that cannot be recovered, or when their outcomes cannot be easily predicted, as discussed above.

3.2.3. The Guess Sub-Pattern

The guess sub-pattern is used to begin the optimistic execution process by making an assumption on the expected outcomes of the execution of a particular action. Once again, in a networked game, this will typically require collaboration between a local entity and a remote entity in a similar fashion as in the caution sub-pattern above. The difference is that in this case, local execution may proceed, thereby masking the latency of remote result computation and communication.

In using this sub-pattern, the guess process must be provided both with the expected results of the action being executed, to be used in later verification, along with a set of recoveries that can be used in case the actual computed results do not match the expected results.

3.2.4. The Padding Sub-Pattern

The padding sub-pattern is used to add some form of distraction element to the game to reduce the amount of optimistic execution that is allowed to occur. This can be used in a wide variety of situations, but is particularly useful when the recoverability or predictability of an action meets the threshold to proceed optimistically (in the optimism decision sub-pattern in Section 3.2.1), but is below a second threshold of comfort and still somewhat questionable as a result.

By employing this sub-pattern, the amount of recovery required is lessened if the original assumption was incorrect, because the amount of optimistic execution was lessened. At the same time, the distraction element in the padding still effectively masks the latency of result computation and communication that is occurring in parallel.

To carry out the padding sub-pattern, a decision is first made as to whether padding is necessary or not. This could involve a similar decision process as that used in optimism decision sub-pattern applied to different thresholds, or an entirely different decision process. This decision process must also determine which methods of padding are appropriate in the current situation and select one 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.) The padding is then executed, and optimistic execution continues upon the completion of the padding. It is important to note that padding may consume either a part or all of the time that would have been spent executing optimistically, 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.)

3.2.5. The Synchronization Sub-Pattern

The synchronization sub-pattern is used to provide synchronization primitives for optimism. Synchronization can be added to either check on the status of a remote computation (non-blocking mode) or to force a wait for the remote computation to complete (blocking mode). This can be used to prevent further optimistic execution from proceeding if that execution would be difficult to recover from. It is important to note that recovery would still be necessary upon denial for any optimistic execution up until this point, however.

To carry out the synchronization sub-pattern, a decision is first made as to whether synchronization is necessary or not. This decision process is, once again, game and action specific. If a decision is made that synchronization is necessary, a synchronization point is used to cause execution to block and wait until the computation of results for the synchronized action is complete. If synchronization is not deemed to be necessary, optimistic execution is allowed to proceed.

3.2.6. The Result Check Sub-Pattern

The result check sub-pattern is used at the end of pessimistic or optimistic execution to collect results from the action initially carried out, and to allow the game to resume normal execution and control flow. How this is done depends on whether pessimistic or optimistic execution was in use.

If execution was pessimistic, the game would currently be blocked waiting for the results of the action invoked to be computed. With the results of the action now in hand, execution can simply continue at this point, keeping the results in mind. While time was lost in the process, nothing special needs to be done to proceed.

If execution was optimistic, there are two possibilities. If the assumption made when using the guess sub-pattern was correct, the assumption is said to be affirmed, and execution can continue at this point with the latency of the underlying computation and communication effectively hidden. If the assumption made was incorrect, however, it is said to be denied. In this case, recovery is needed to cope with the incorrect optimistic execution completed in the mean time, to bring the game back into an acceptable state.

3.2.7. The Recovery Sub-Pattern

The recovery sub-pattern is used to bring a game back into an acceptable state following the denial of an optimistic assumption. Since multiple recoveries may be possible, a recovery selection procedure must be followed to determine the best recovery the handle the current situation. After the execution of this recovery, the game is allowed to proceed from this corrected state.

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.

3.2.8. Additional Elements

In addition to the above core elements to the overall optimism pattern, there are several other elements that can be applied. Two of these are discussed below; for further details, the reader is again urged to consult [18].

One additional element is the pattern of nested optimism. The overall pattern of optimism presented in Figure 1 depicts the flow of control for a single optimistic assumption. The situation quickly grows more complex when additional optimistic assumptions are made as part of the optimistic execution of the original assumption. This results in a nested optimism, in which the denial of one assumption can lead to a cascading denial of all optimistic assumptions and require a complex recovery procedure. Because of the complexity involved with nested optimism, its support is not required, but can be quite useful in some gameplay situations.

Another element is an optimism feedback pattern. In this case, feedback from result checks is applied to earlier decision processes to provide positive or negative reinforcement of decisions depending on the affirmations and denials of optimistic assumptions. In this way, the optimistic execution process can be tuned and adapted automatically during a game's execution. Again, since this can be complex to support, it is not a required feature.

4. Case Study in Using Patterns of Optimism

As proof of concept, the Football Invaders football game was developed using the optimism patterns described in Section 3. The design of this game was based on a modification of a single-player version of Space Invaders. All of the graphics were acquired from Cinemaware's TV SPORTS: Football. The game itself was written in Java. A screenshot of this game is shown in Figure 2.

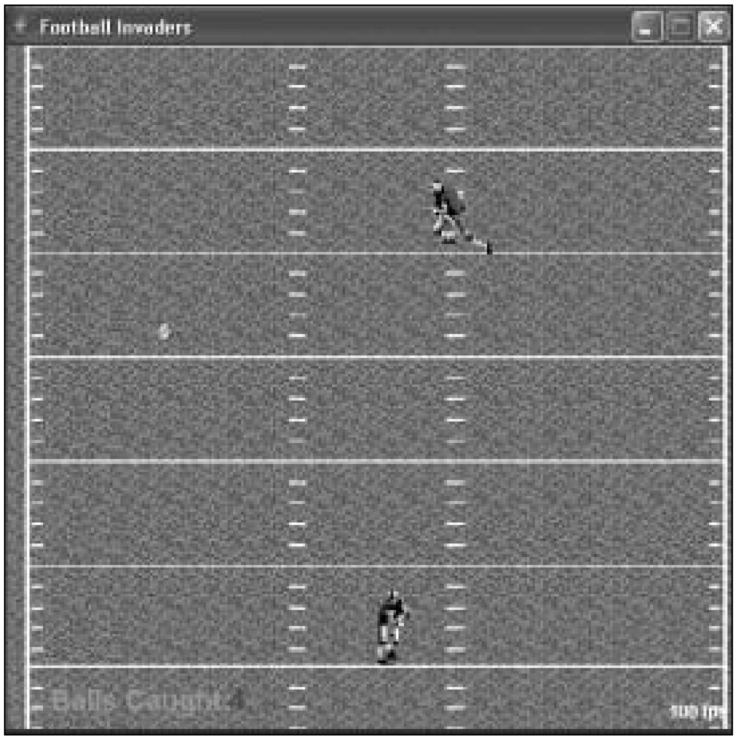

Figure 2. Screenshot of Football Invaders.

Within Football Invaders, the player-controlled receiver is the football player at the top of the window in Figure 2. The receiver is allowed to move in all directions around the field but cannot come with 15 yards of the quarterback. The computer-controlled quarterback throws a football across the field at regular intervals in a random direction. A wind force is also applied to act upon the football in every direction except the direction of the quarterback. The goal of the game is to catch the football in light of the unpredictable wind (and quarterback).

Football Invaders uses a client-server architecture, with UDP 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 client and the behaviour of other game entities. This includes calculating new wind speeds and updating the football's position accordingly. This updated game state is then sent back to the client. At the client, updated game states related to the last player input are rendered to the display as they are received.

After Football Invaders was developed, our patterns of optimism were used to create a new set of Java classes for assumptions, recoveries, paddings, and so on, in addition to an optimism class to drive and manage optimistic execution. While these classes were logically separate, they had to be aware of much of the inner workings of Football Invaders in order to effectively manipulate its execution as necessary. Consequently, the implementation of our patterns of optimism in this case was tied specifically to Football Invaders, and was not portable. In the future, we plan to investigate ways of having more general, flexible, and portable optimism code constructs that can be used without modifications in other games. Our earlier work in [12], however, indicated that this will be a difficult task indeed.

Our optimism patterns were specifically used in Football Invaders to make optimistic assumptions about the catching of footballs. If the client detected that the player-controlled receiver was within a certain distance from the football with appropriate wind conditions, the client would optimistically assume that the receiver caught the football while waiting for a confirmation update from the server. Optimistic execution in the mean time would show the player catching the ball and continue moving. Upon affirmation of the optimistic assumption, the game would simply proceed as normal. If the assumption was denied, meaning that the player did not, in fact, properly catch the football, the player bobbles and drops the ball instead, as shown in Figure 3.

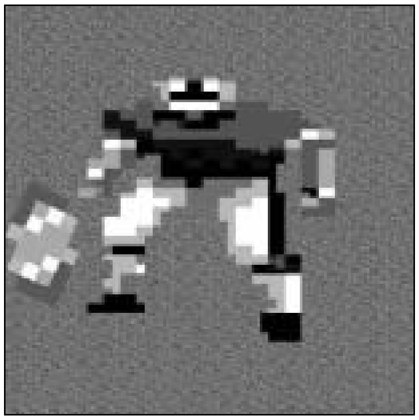
Figure 3. Screenshot of Recovery in Football Invaders.

Padding was also added to Football Invaders. When the player-controlled receiver was in range to catch the football and trigger optimistic execution, but at the outer edge of this range, padding would execute. This padding would slow the game on the client and show a zoomed in view of the player, giving the server sufficient time to produce and send an update to the client indicating whether the ball was actually caught or not. This is shown in Figure 4. Synchronization was also used at the end of optimistic execution to prevent scoreboard updates until after an affirmation update from the server.

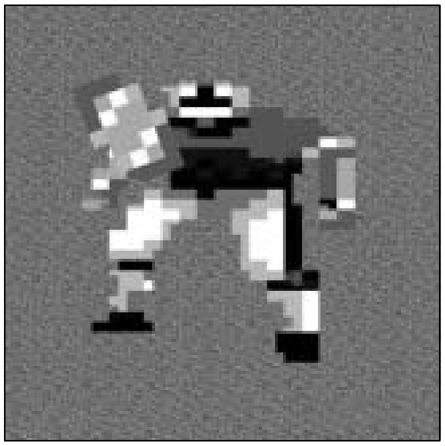
Figure 4. Screenshot of Padding in Football Invaders

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

Based on these results, it is expected that other developers can use these patterns to add optimistic execution to networked multiplayer games successfully. Consequently, these patterns of optimism could prove quite useful to reducing the effects of latency in games.

5. Conclusions and Future Work

Latency remains a challenging problem to the development and success of networked multiplayer games. New HOPE is aimed at reducing or eliminating the effects of latency to produce more enjoyable gaming experiences for players. Through the patterns of optimism introduced 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 design patterns in the development of a simple football game, Football Invaders, has shown their usefulness, and demonstrates great promise for the future.

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

- Further experimentation with our patterns of optimism and Football Invaders is clearly necessary. We need to fully investigate the latency reduction benefits of optimism in this game, and learn how to further tune the factors influencing optimism decisions to improve performance. Initial tests have found that simple adjustments can have pronounced effects on gameplay [18], and so further study is warranted.

- We also plan to have these patterns of optimism used in the development of additional games, constructed by other developers. This will validate their use and provide valuable feedback for their refinement.

- Further study is also required into the use of both nested optimism and optimistic feedback. Neither of these sub-patterns was used in the development of the initial prototype of Football Invaders, and so implementation and experimentation efforts are currently under way.

- Many of approaches to latency compensation discussed in Section 2, including dead reckoning and so on, have predictive elements that, in the end, make them quite similar to the constructs used in optimistic execution. Consequently, in the future, we plan to use the patterns of optimism introduced for New HOPE 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.

- Based on our on-going experiences with optimistic execution and New HOPE, we plan to continue investigating the feasibility of developing software APIs and libraries to support optimistic execution within networked games. This is a particularly challenging prospect considering the diversity of gameplay elements found in different games, but could produce large benefits at the same time.

References

1. S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan. "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 2004.
2. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, New York, 1977.
3. G. Armitage. "Sensitivity of Quake3 Players to Network Latency". *Presented at the SIGCOMM Internet Measurement Workshop*. San Francisco, California, November 2001.
4. J. Aronson. "Dead Reckoning: Latency Hiding for Networked Games." *Appeared in Gamasutra. Available at http://www.gamasutra.com/features/19970919/aronson_01.htm.* September 1997.
5. Y. Bernier. "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization." *Presented at the 2001 Game Developers Conference*. San Francisco, California. March 2001.
6. S. Björk and J. Holopainen. *Patterns in Game Design*. Charles River Media. 2005.
7. J. Blow. "Miscellaneous Rants". *Appeared in Game Developer Magazine*. May 2004.
8. R. Carlson, T. Dunigan, R. Hobby, H. Newman, J. Streck, and M. Vouk. "Strategies & Issues: Measuring End-to-End Internet Performance". *Appeared in Network Magazine*. April 2003.
9. C. Cowan. "A Programming Model for Optimism". PhD Thesis. Department of Computer Science, The University of Western Ontario. February 1995.
10. J. Fraser. "Zeroping Frequently Asked Questions". *Accessible online at: http://zeroping.home.att.net.* April 2000.
11. E. Gamma, R. Helm, R. Johnson, and J Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 1995.
12. R. Hanna and M. Katchabaw. "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 2005.
13. L. Pantel and L. Wolf. "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 2002.
14. L. Pantel and L. Wolf. "On the Suitability of Dead Reckoning Schemes for Games". *Proceedings of the First Workshop on Network and System Support for Games*. Bruanschweig, Germany, April 2002.

15. M. Mauve. “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 2000.
16. PricewaterhouseCoopers LLP. Global Entertainment and Media Outlook: 2005-2009. *PWC Report*, 2005.
17. J. Schirra. “Content-Based Reckoning for Internet Games”. *Proceedings of the Second International Conference on Intelligent Games and Simulation (GAME-ON 2001)*. London, England, November 2001.
18. G. Shelley. “Design Patterns for Optimism in Networked Games”. *Masters Directed Studies Report. Department of Computer Science, The University of Western Ontario*. In Progress.
19. J. Smed, H. Niinisalo, and H. Hakonen. “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 2004.