Quantifying the Effect of Latency on Game Actions in BZFlag

Latency and Games: Measuring the Effects of Latency on Gameplay in an Online Game

A Major Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

By:

Christopher Burgess

Nathan Roy

Date: March 4, 2009

Report submitted to:

Professor Mark Claypool

Abstract

Latency's effect on performance in online games is becoming an increasingly relevant area of research. While much research has been performed on the broad topic of latency in games, this project studied the effect of latency on game actions based on their precision and deadline characteristics. We modified a publically available game to allow for artificially induced latency, ran experiments in that game with varying precision and deadline characteristics, and analyzed the resulting data to determine the trends.

Contents

Chapter 1: Introduction

Online games, which are video games where players play with or against each other over the Internet, are growing in popularity throughout the world. For example, Blizzard's *World of Warcraft* has gathered over 10 million subscribers throughout the world since its launch in November, 2004 [1]. In order to provide a better experience for the growing number of users playing these games, many researchers have turned their focus towards issues affecting online game performance, such as network latency.

Latency is the delay between when an action is initiated by a game client and when it is physically recognized by the game server. There have been many studies into the effects of latency on user performance in commercially available games such as Unreal Tournament 2003 [3], Madden [4], and Warcraft 3 [5]. In these studies, the effects of latency on player performance were evaluated based upon the player's overall scores. While the Unreal Tournament 2003 study did categorize weapons based on precision required, it did not study the game actions as a whole, which left out actions such as running and jumping and did not consider the deadline required for each weapon. These studies were effective in showing the effects of latency and packet loss on individual player performance, but their results cannot be easily translated or quantified based on the precision and deadline characteristics of game actions.

While latency can be detrimental to a game's performance, not all actions within a game are equally sensitive to latency. Research has been done to provide criteria for which game actions can be judged based on two properties: the action's required precision and deadline [2]. The precision of a game action is the accuracy which is necessary in order to successfully complete the action. An example of precision is the size of a target within the player's crosshairs. The deadline of an action is the time required in order to successfully achieve the desired outcome. An example of deadline is the time a target remains in the player's crosshair before going out of range. The goal of our project is to determine whether or not actions in games are affected differently by latency based upon their precision and deadline requirements. For example, actions with high precision requirements and short deadline, such

as shooting an enemy in the head from long distance will be more adversely affected by latency then the act of shooting a cannon ball at an enemy ship.

Although there has been significant research relating to latency in games, to the best of our knowledge, there has not been any research done into the effects of latency on specific game actions based on their precision and deadline characteristics. Research into the effects of latency on game actions with specific precision and deadline requirements can help game developers understand what actions need to be optimized for network performance and which actions do not depend so heavily on high network performance.

Our study aims to use a publicly available game called *BZFlag* (http://*BZFlag*.org/) to study the effects of latency on specific game actions which have been categorized by their precision and deadline requirements. *BZFlag* is an online multiplayer tank shooter game where players play as a tank and are tasked with eliminating other players. The game can also be played by robot players, or bots, which were used extensively during our experiment. The reason we used bots for our experiments was to eliminate the factor of player skill differences on our final results. In order to use *BZFlag* for our experiments, we first modified the game's code to allow us to induce specific latencies on the players in the game. We also modified *BZFlag's* code in order to allow us to change tank size and bullet speed, which provided us control over the precision and deadline characteristics of the game's main action, shooting. Lastly, we modified the bot's AI code to make them more selective Next, we ran tests using our modified *BZFlag* game with eight bots and varying levels of induced latency to study the effects of latency on actions with varying precision and deadline requirements.

After running these tests, the results obtained support the claim that game actions are affected differently by latency based on their precision and deadline characteristics. We ran tests in the precision space with varying latencies and found that in games run with higher precision requirements, a tank with latency performed worse than games run with lower precision requirements. Next, we ran tests in the deadline space and found that in games run with shorter deadline requirements, a tank with latency performed worse than games with longer deadline requirements. Lastly, we ran tests with variances in both the precision and

deadline spaces and found that in games with high precision and short deadline requirements, the tanks with latency performed worse than tanks in a game with low precision and long deadline requirements. The results are discussed in more detail in Chapter 5.

The rest of this paper is organized as follows: Chapter 2 presents background information and the discussion of related work. Chapter 3 details our approach and experiment design for the study of latency on game actions based on their precision and deadline characteristics. Chapter 4 details the implementation and design of the modification made to the *BZFlag* source code. Chapter 5 discusses the results of the experiments outlined in the approach chapter. Lastly, Chapter 6 discusses the implications of the results of our experiments with regards to other online games.

Chapter 2: Background / Related Work

Other attempts to examine the effects of latency in multiplayer online games have focused primarily on user studies. Some examples of games studied include Unreal Tournament 2003 [3], Madden NFL Football [4], and Warcraft III [5]. These user studies generally focused on a single game or an aspect of a game, and consisted of various users playing the game both without latency and with latency added. The experiment administrators would then measure the difference in player performance as latency increased based on the changes in the player's score. As is to be expected, most of the referenced studies have found that player performance degrades as latency increases; however, certain types of game actions are more resistant to performance degradation due to latency, whereas others are more susceptible to latency. What these user studies generally found was that actions requiring high precision (like shooting a sniper rifle at a faraway target) or that require a very quick response (like making a sharp turn in a racing game) are more affected by latency than actions requiring less precision. While user studies can provide definitive and quantifiable results as to the effects of latency in the game they focus on, it is often difficult to apply these results and findings to other games. Another factor with user studies is the skill of the players playing the game; this can be difficult to measure exactly. Our study attempts to provide a model for the effects of latency on various actions in a game that can be applied to other games as well.

The variation in the results of latency tests performed on games indicates that there are different categories of actions that can be executed in a game. The different categories of game actions will also be affected to different degrees by network latency. It has been theorized [2] that the susceptibility to latency of a particular action can be categorized based on two attributes: the *precision* with which the action must be executed in order to be successful, and the *deadline* by which the action must be performed by. Actions with higher required precision and shorter deadline requirements are more susceptible to latency [2]. Therefore, actions that require both speed and accuracy are the most difficult to perform in a game environment. Actions that require either accuracy or quick response time, but not both, are affected by latency as well; however, the user can compensate partially for the slow network response time. Finally, those actions that have no precision or deadline requirements are

hardly affected at all. The user studies support this hypothesis; games such as first-person shooters, which require quick and precise actions, are the most adversely affected by latency, whereas real-time strategy or turn-based games are the least affected. Games of the sports and role-playing genres, which generally have moderate precision requirements, are moderately affected by latency. The precision-deadline model of categorizing player actions can give some insight as to how susceptible a particular game or aspect of a game will be to latency; however, it too falls short of being able to quantify the difference, as the changes in performance requirements from one game to another are difficult to evaluate. Also, even within a specific game, there are many different actions which can be affected by latency to varying degrees which is something the previous studies have failed to take into account. Our study aims to categorize specific actions in a game and demonstrate the degree to which they are affected by latency.

Research has also been done into the effectiveness of latency compensation techniques, such as dead reckoning, which minimize the impact of latency on gameplay. One such attempt [6] utilizes a simulation tool known as GLS (Game Latency Simulator) to test the effects of various latency compensation techniques. GLS is a simplified version of a first-person tank shooter game, called BZFlag, which pits a set number of computer-controlled tanks against each other with the goal being to shoot as many of the opposing tanks as possible. The simulator also has the capability to emulate network latency, and this delay can be adjusted by the user as the simulation runs. Unlike most actual games, GLS also allows for modifications to tank speed, bullet size, and shot speed, among other factors. The user can also choose which form of latency compensation the computer-controlled tanks will use. This customization makes GLS a useful tool for evaluating latency compensation techniques, but it has other configurable features which make it useful for game latency studies. GLS's ability to easily adjust game mechanics can also be used to adjust the precision required to hit another tank, as well as the deadline by which to do so. This provides a method to evaluate the effects of precision and deadline on performance degradation without having to look at different games in order to do so. However, GLS is very limited in what it can do; each simulation has only two tanks, latency can only be applied to both tanks, and it also did not track the hits the tanks

scored on each other. For our study, we needed to be able to include more than two bots, be able to induce latency on specific tanks in the game, and also needed to track scores as well as other statistics.

The Game Latency Simulator is, as stated above, loosely based on an open source tank shooter game called *BZFlag*. Like the simulation based on it, the game *BZFlag* is straightforward in nature – the goal is simply to control a tank and shoot as many other tanks as possible. Players can also enhance their tanks' capabilities by picking up “superflags”, each of which grant a specific bonus, such as increased rate of fire or increased turning speed. Due to the simple nature of the game, it is relatively easy to design computer-controlled players to play the game and do so with a reasonable measure of efficiency. The computer players in *BZFlag*, while not truly comparable to a skilled human player, are evenly matched when pitted against each other, and therefore offer a testing medium that eliminates the human skill aspect of a user study. These factors make the use of *BZFlag* attractive for our study.

Chapter 3: Approach

3.1 Possible Approaches

We researched three games in which to perform our study of the effects of latency on game actions which have been categorized by their precision and deadline requirements. The first option was to use a fully commercially released first person shooter game called *Quake 3* (http://www.idsoftware.com/games/quake/quake3-arena/). The second choice was to use a game simulator called GLS (Game Latency Simulator). The final option we had was to use the publicly available tank shooter game called BZFlag. Out of all of these options, we were tasked with deciding which choice would provide the best balance between ease of modification and relevant results.

Our first test bed option, *Quake 3,* is a commercially available game whose source code was recently released by the game's developer, ID Software. Quake 3 is a multiplayer online first person shooter game which is based on ID Software's ID Tech 3 engine. The game features the ability to play with robot players (bots for short), client-server architecture, and the ability to extend the game's code to produce complex modifications (or MODs).

While the use of a commercially available game, such as *Quake 3*, would have made the results of our study more interesting and relevant to a larger community, it is based on a very complex code base which would have required a lot of modification in order to work for our study. *Quake 3* does not have the ability to artificially induce latency per player, and adding such functionality to a complex code base would have proven difficult. Also, modifying the game's bots to use specific weapons which have been analyzed based on their precision and deadline requirements would have to have been implemented. For these reasons, we decided to not use *Quake 3* for our study of the effects of latency on game actions which have been categorized by their precision and deadline requirements.

An alternative to using *Quake 3* was to use the Game Latency Simulator (GLS). GLS, which is based on the popular game *BZFlag*, is on the opposite end of the spectrum from *Quake 3* as far as market proliferation is concerned. GLS was developed by a group of researchers for

the specific purpose of measuring the effectiveness of certain latency compensation techniques. Since GLS is not a commercially available game, it is hard to judge the quality of the results and they would be difficult to apply to a more general category of commercial games. Also, GLS would require some modifications in order for it to provide the functionality required for our study. These changes include modifying GLS's latency code to allow latency to be applied to each player individually, and also adding code to keep track of and log player performance statistics such as hit percentage. However, the game's code base is far less complex compared to that of *Quake 3,* and the required modifications would not be as difficult to implement. Although GLS remains a suitable option for our study, we decided to use a more ubiquitous and flexible game in our study.

The game we decided to use for our approach was the publicly available game called *BZFlag*. *BZFlag* is still relatively popular in the online community, with our measurements showing an average of 20 active servers (servers with 2 or more players connected) at an off peak hour (11 am EST). Stats regarding the number of servers and players currently playing are continuously being tracked online (http://stats.bzflag.org/). *BZFlag*, which is the game that GLS is based on, provides far more features than GLS does. Also, due to the fact that it allows human players to participate it affords us the opportunity to perform a user study if time permits. The use of a relatively popular online game such as *BZFlag* should provide our study's results with more credibility and relevance to possibly interested researchers. *BZFlag* still required two major modifications to the game's code base; adding the ability to induce latency to the network code and a minor change to the code to allow the game to use a configuration file to determine tank speed and bullet radius. However, due to the relative simplicity of *BZFlag*'s code, the necessary modifications were not expected to be a major undertaking.

3.2 Experiment Design

Once we completed the necessary modifications to the *BZFlag* source code, we were able to begin running tests to determine the actual effects of our artificially-induced latency on in-game action. Our first goal in this regard was to define what constituted a test run of our experiment, as well as the control values for that test run, both with and without latency. Once

we established normal values for the statistics we planned to measure by running an unmodified game of *BZFlag*, we then proceeded to modify the precision and deadline requirements of the game's basic action of shooting at and hitting an opposing tank.

For our default experimental run, we had eight computer-controlled players, divided into four teams of two, compete against each other for a predetermined amount of time. This time was determined by running many tests of different length with eight computer players and using the time frame which produces the most even results in the shortest amount of time. Eight computer players is optimal given the size of the default map in *BZFlag* – it gives us a good number of test subjects to measure results without making the map too crowded. Also, having eight players for the test runs is optimal because it is a standard number of players in many first person shooter game tournaments [8]. Once we have these results, we induced latency to one of the computer controlled tanks and measure its performance relative to the other players over the same time period as our control run. This showed the effect of latency on the default actions of BZFlag.

Once the effect of latency on performance has been established, the next step was to make a modification to some aspect of the game that changed either the precision or the deadline required to score a hit against another tank. To measure precision, we modified the size of the tanks. A larger tank size reduces the precision requirement for a successful shot because the size of the target is larger. To measure the effects of latency on deadline requirements, we modified the speed of the bullets that the tanks fire. Modifications which provide a faster bullet speed afford players more time to plan a shot before firing, thus affecting the deadline of the action. Once tests were run on the precision and deadline spaces individually, tests involving modifications to both simultaneously were run. We were most interested in the extremes of the precision and deadline spaces, so the tests involved the values at the boundaries of either space.

The above experiments gave us a high quality sampling of statistics on which to base our analysis. The most basic performance measurement in *BZFlag* is the player's score, which is the difference between number of kills and number of deaths. The computer-controlled players

are evenly matched in the latency-free experiments, so if they are allowed to continue to play for a long enough period of time, they generally accrue scores close to 0. This period of time was used for all of the experiments in order to provide data with reasonable deviation. The score of a player affected by latency in a particular experiment was the main focus of our test and it gave us a good indication of the level of performance degradation latency causes under those circumstances. Another statistic we monitored is the player's hit percentage. This is, in effect, the success rate of the action being performed, and therefore the easiest to quantify and compare to other games. A player with latency is expected to have a lower hit rate, but these experiments helped us determine how hit rate is affected by modifications to precision and deadline characteristics. Finally, the number of shots fired is a statistic that was also affected by these same factors, as players affected by latency do not have as many opportunities to take shots.

In order to perform the experiments required, we first had to make modifications to the *BZFlag* source code. The design and implementation details of these required alterations are detailed in the next chapter.

Chapter 4: Implementation

In order to test our hypothesis, we first had to design a way to introduce latency in a controlled fashion into a local (robot player) *BZFlag* match. To do so, we chose to modify the source code for *BZFlag*. Also, in the course of our original tests, we found that specific Artificial Intelligence characteristics were giving lagged tanks an advantage in some cases; we implemented changes to the AI code in order to address this problem. In this chapter, we will discuss our code design and implementation for the artificial latency modification and the AI changes.

4.1 Code Design and Implementation for Latency Modification

In order to add artificial latency to the CPU controlled players (henceforth referred to as *bots*) of *BZFlag*, we first had to deduce a proper design which could easily and flexibly achieve our desired functionality. During our study of the *BZFlag* source code, it became obvious that the time keeping code which was included would not provide sufficient granularity for any latency inducing modifications and that we would have to first write our own. Fortunately, we were able to find time tracking code for the Windows platform on the Web [7] which we were able to include in our modifications to the *BZFlag* source code.

The design we decided to employ was to create a new class called the `LatencyManager` which is responsible for acting as a proxy between the bots and the server, delaying messages sent for the bots which require artificial latency. The `LatencyManager` class does this in two general steps.

After an initial setup, the first step of the process occurs whenever a bot tries to send information to the server. Before the information is sent, the `LatencyManager` is first queried to see if latency should be added to the packet. If so, the LatencyManager then stores that packet's required information into a `DelayedMessage` structure and queues that structure to be sent at a later time. This process is done in the `LatencyManager::addDelayedMessage()` method which is detailed in Figure 4.1.

```
void LatencyManager::addDelayedMessage(uint16_t code, uint16_t len, const
void* msg, PlayerId id)
{
	timeval t;		//timestamp structure
	/* create a timestamp which represents the time to send the msg*/
	this->createTimeStamp(&t, tank_latencies[id]);

	/*create a DelayedMessage with the proper info */
	DelayedMessage dl;
	dl.code = code;
	dl.len = len;

	/* copy the message and info into the structure */
	dl.msg = (void *)malloc(len);
	memcpy(dl.msg, msg, len);
	dl.player_id = id;
	dl.timeToDeliver.tv_sec = t.tv_sec;
	dl.timeToDeliver.tv_usec = t.tv_usec;
	dl.sent = false;
	/* queue the message */
	messages.push_back(dl);
}
```

Figure 4.1: LatencyManger::addDelayedMessage() – Queues a packet to be sent later.

In order for the first step to be performed modification of the *BZFlag* `ServerLink::send()` method was required to accommodate the additional logic. The `ServerLink::send()` method was chosen because it is the single function that every sending method calls in order to send data to the server. Therefore, the original contents of the `ServerLink::send()` method were moved to a new method called `ServerLink::doSend()` and new code was introduced which determined if a packet should have additional latency. The new `ServerLink::send()` is detailed in Figure 4.2. In this code, only the messages for the tank to begin shooting are lagged. The reasons for this are detailed in Section 4.2.

```
void              ServerLink::send(uint16_t code, uint16_t len,
                                          const void* msg)
{
      //* Only lag the shots of tanks to accurately simulate lag
      if((LatencyManager::instance().getLatencyForTank(this->getId()) != 0)
&& code == MsgShotBegin)
      {
            LatencyManager::instance().addDelayedMessage(code, len, msg,
this->getId());
      }
      else
            doSend(code,len, msg);
}
```

Figure 4.2: ServerLink::send() – Diverts all packets through the LatencyManager if the tank has a nonzero value for latency.

The second step in the process is initiated every game update, where the `LatencyManager::Update()` method is called. Game updates happen approximately every 15ms. In this method, the `LatencyManager` determines if any of the packets in the queue are ready to be sent, and if so, the `LatencyManager` calls the appropriate bot's `ServerLink::doSend()` method. The `LatencyManager::Update()` method is detailed in Figure 4.3.

```
void LatencyManager::Update()
{
	/* get the current timestamp */
	timeval t;
	this->getTimeOfDay(&t, NULL);
	/* iterate through all of the queued messages */
	vector<DelayedMessage>::iterator it = messages.begin();

	it = messages.begin();
	for (it; it != messages.end(); it++)
	{
		/* if the message is ready to be sent */
		if(timeStampHasExpired((* it).timeToDeliver, t))
		{

			/* send the message and mark the message as sent so
			it can be pruned */
			getServerLinkForTank((*it).player_id)->doSend((*it).code,
						(*it).len, (*it).msg);
			(* it).sent = true;
		}
	}

	/* remove the sent messages from the queue */
	messages = pruneSentMessages();
}
```

Figure 4.3: LatencyManager::Update() – Checks to see if the latency time has elapsed for each stored packet, and sends the packet if time has elapsed.

Through the implementation of the LatencyManager, we were able to artificially induce latencies on specific bots in *BZFlag*. Code testing results and performance data will be explained in the next section.

4.2 Validation of Latency Modification

During the testing of the artificial latency code, we employed log files and timestamps in order to ensure that desired effects were achieved.

Through our testing efforts, we were able to tune and fix our code in order to obtain our intended final product, which was a game in which we can induce controlled amounts of latency to specific players. Originally, all packets sent from a lagged bot would be queued in the `LatencyManager`, therefore adding latency to everything the bot does. However, during testing we found that *BZFlag* collision detection is handled at the client, which is unusual. Traditional first-person shooters with a client-server architecture handle collision detection at

the server. This meant that the difficulties a lagged tank experienced in successfully hitting other tanks also applied to the other tanks attempting to hit the lagged tank. This was true because of the client-side collision detection mentioned above. Since the other bots were not receiving updates on the current location of the lagged bot, they were unable to shoot it accurately. In order to address this situation, we decided to lag only shooting messages from the lagged bots. Delaying only the shot messages allows for all of the other bots to react to the lagged bot's position (which emulates how most games keep track of positions server-side) and therefore be able to shoot at it normally, while the lagged tank has the disadvantage of having its shots delayed, leading to more misses.

Verification of the desired effects of our artificial latency code can be shown through the log output during the game's execution. A snippet of the log output is detailed in Figure 4.4.

```
--------------- Game Started at: 2009-01-23 10:21:16 -----------------------
Latency of 10000 added to : Nate00 (ID: 1)
...
Packet Queued around: 1232724077 953125
SENDING MESSAGE FROM: 1 2009-01-23 10:21:17
Expected Lagged Deliver Time: 1232724077 963125
Actual Lagged Deliver Time: 1232724077 968750
```

Figure 4.4: Log Output from Sample Run

Analysis of the log output shows that a latency of 10000 usec (10 ms) has been added to bot with ID 1. The next message shows that a packet was queued from ID 1 at **1232724077** seconds and **953125** usec. The next message shows that a packet from ID 1 was actually sent at **1232724077** seconds and **968750** usec. This is a difference of **15625** usec (about 15 ms), showing that there is actually at least 10 ms latency being added to this packet. The extra time being added to the latency is due to the update rate of the game, which happens every 15ms, and will be negligible in our tests at higher latencies.

Before testing the latency inducing code in a live game, we first performed a six hour control test with two un-lagged bots. This was to verify that the bots perform equally well against each other in the game's default settings. After the designated time period was over, each bot finished with extremely similar kill death ratios as well as hit percentage, as we

expected. Each of the bot's hit percentage remained around 25% and each bot had a nearly even kill to death ration, with a variance of 6 kills/deaths. Figure 4.5 shows the results of the 6 hour control test. These results demonstrate that over a lengthy period of time, there is no appreciable performance difference between two unmodified bots. This means that when latency is introduced, any performance decrease that results can be attributed to the effect of that latency.

Player	Score	Kills	Deaths	Hit %
Tank 0	-6	640	646	24%
Tank 1	6	646	640	25%

Figure 4.5: Results of 6 Hour 1v1 Control Test

Second, we tested *BZFlag* with one normal bot and one lagged bot in order to determine if there was in fact performance degradation due to the added latency. This simulation was run for 12 hours, in order to definitively eliminate variance due to length of test run, and the lagged bot finished with noticeably poor results of a score of -674 and a hit percentage of 26% when compared with the non-lagged bot's score of +674 and hits percentage of 32%. Figure 4.6 shows the results of the 12 hours latency test.

Player	Score	Kills	Deaths	Hit %
Tank 0	-674	1087	1761	26%
Tank 1	674	1761	1087	32%

Figure 4.6: Results of 12 Hour Latency Test

The results of the aforementioned tests suggest that the artificial latency code that we added to the source code of *BZFlag* works as intended. Upon assurance that the modifications to *BZFlag* work as anticipated, we were able to continue on and gather data through running the experiments laid out in our test plan.

4.3 AI Aiming Modification

During the course of our original experiment runs, we found that in certain situations, particularly those with low precision and deadline requirements, lagged tanks actually were

shown to have an advantage over non-lagged tanks. The experiments in which this phenomenon occurred were when bullet speed was reduced to ½ or ¼. The advantage was a result of the fact that non-lagged AI tanks were choosing to shoot when their chance to hit was not great enough. The non-lagged tank's shot would miss and the lagged tank's shot would then be sent, effectively causing it to fire with a reduced range, increasing its chance to hit. This was most prevalent in situations where two enemy tanks were approaching each other head-on, which is what the AI is programmed to do. The reason this was a problem is because in *BZFlag,* a tank may only fire one bullet at a time, and must wait until the previous shot expires before firing again, which takes 3.5 seconds.

We were able to track the problem down to a behavior in the Artificial Intelligence code for determining when to shoot. The original AI code, shown in figure 4.6, determined that a good shot was when the tank's range to miss their target was not more than one half of the target tank's length.

```
const float missby = fabs(azimuthDiff) *
      (targetdistance - BZDBCache::tankLength);
    // only shoot if we miss by less than half a tanklength and no building
    if (missby < 0.5f * BZDBCache::tankLength &&
      p1[2] < shotRadius) {
```

Figure 4.6: Original AI Code for Shooting

The constraint of not missing by more than one half of the target's length was found to be too imprecise, resulting in a tendency to miss on the initial shot when approaching another tank. In order to determine a more appropriate choice for this constraint, we tested multiple different values. The results of these tests are shown in figure 4.7. For these tests, the tanks' bullet speed was reduced to ¼, and one tank was lagged by 1000ms because the aforementioned problem occurred most at these settings.

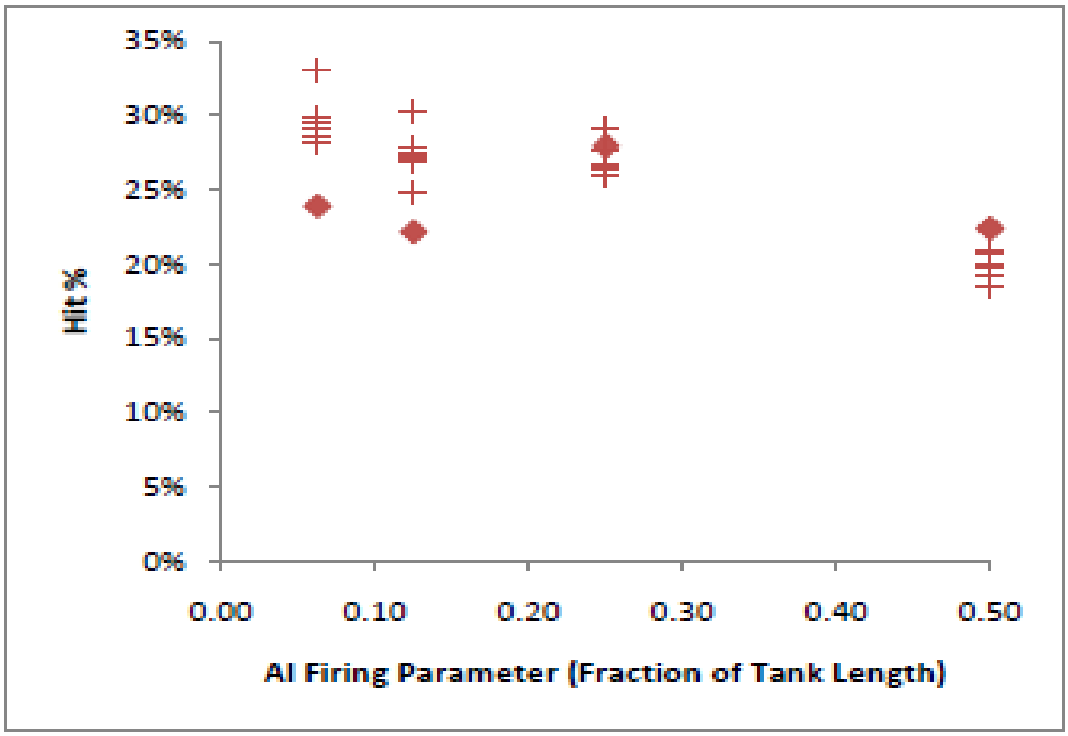

Figure 4.7: Results of Varying Miss by Value Tests

The graph in figure 4.7 shows the effects of modifying the bots' AI in one of the situations where the lagged tank had a slight advantage. The X axis shows the fraction of a tank-length the shot must be expected to come within hitting another tank by before the tank will shoot. The Y axis shows the hit percentage. The lagged tank is depicted by a diamond, and the other tanks in the experiment with crosses. It shows that as the tank AI is modified to be more selective with its shots, the overall performance of the tank improves, and the performance of the lagged tank decreases relative to the others. This was determined to be most effective when the fraction of tank length was set to one eighth. As a result, the AI shot determination code was modified to change the fraction to one eighth; this change is detailed in Figure 4.8.

```
const float missby = fabs(azimuthDiff) *
      (targetdistance - BZDBCache::tankLength);
    // only shoot if we miss by less than half a tanklength and no building
    if (missby < 0.125f * BZDBCache::tankLength &&
      p1[2] < shotRadius) {
```

Figure 4.8: Final Modification to AI Code

The aforementioned change to the Artificial Intelligence code in conjunction with the Artificial Latency modification were the major changes added to the *BZFlag* source code for the purposes of our experiments. All subsequent results have these code modifications unless otherwise indicated.

Chapter 5: Results

5.1 Experiment Parameters and Setup

Once the required modifications to the *BZFlag* source code were implemented, we were able to begin the testing phase of the project. All experiments were run on one of two machines. The specifications for each machine are detailed in Figure 5.1.

	Hardware A	Hardware B
CPU	Intel Core 2 Duo T5550 @ 1.83 GHz	Intel Core Duo T2500 @ 2.0GHz
RAM	2 GB 667MHz DDR RAM	1 GB 667MHz DDR2 RAM
Video	NVIDIA GeForce 8400M GS 256 MB	ATI Mobility Radeon X1600 512MB
Resolution	1440 x 900	1200 x 800
OS	Windows XP	Windows XP

Figure 5.1: Hardware Specifications

Each machine satisfied the minimum system specifications required to run *BZFlag* [9]. In order to determine if the difference in hardware was negligible in the experiments, control tests were run and results the results were compared. Each test was run for two hours with eight tanks, organized into teams of two, with no latency added. Results of those tests are detailed in Figure 5.2.

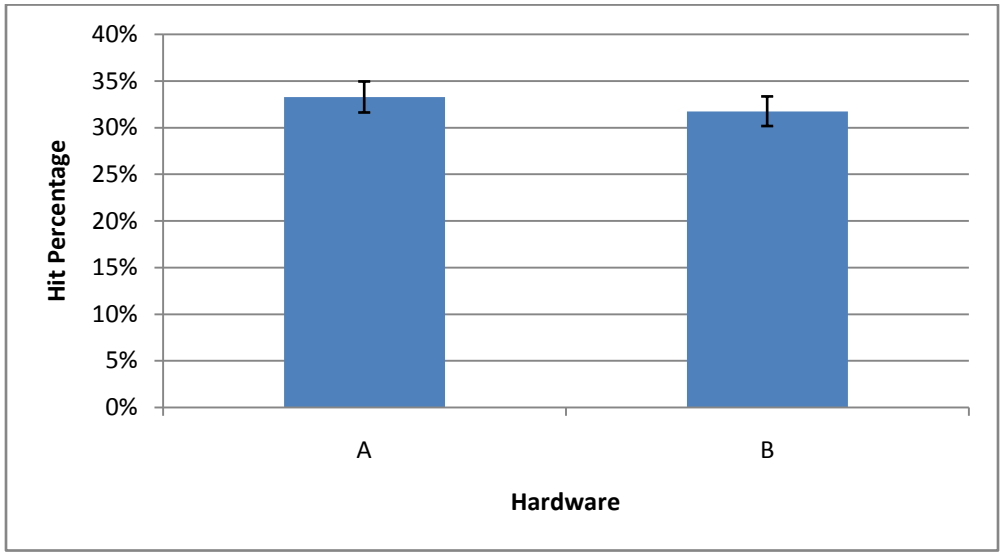

Figure 5.2: Average Hit Percentage vs. Hardware

Figure 5.2 graphs the average hit percentage on the y axis vs. hardware on the x axis. Also noted are the 95% confidence intervals for both of the hardware platforms (denoted in black). The results presented in Figure 5.2 show that there is no significant difference in results based on which hardware the test was run on. Discrepancies in the data can be attributed to random map generation in *BZFlag*. The reason random map generation was a factor is that the randomly placed buildings change the overall layout of the maps, thus creating areas with varying degrees of openness, effectively determining the range at which the tanks can engage each other.

We also needed to determine the length of time for which to run each experiment, while minimizing score variance and the length of the test. Therefore, we ran experiments with the goal of reducing the variance of the bot's scores as much as possible, while also minimizing the amount of time each test would require. In order to do this, we ran multiple tests of differing length, and compared the variance in hit percentage of the eight tanks in the experiment. The results are shown in Figure 5.3.

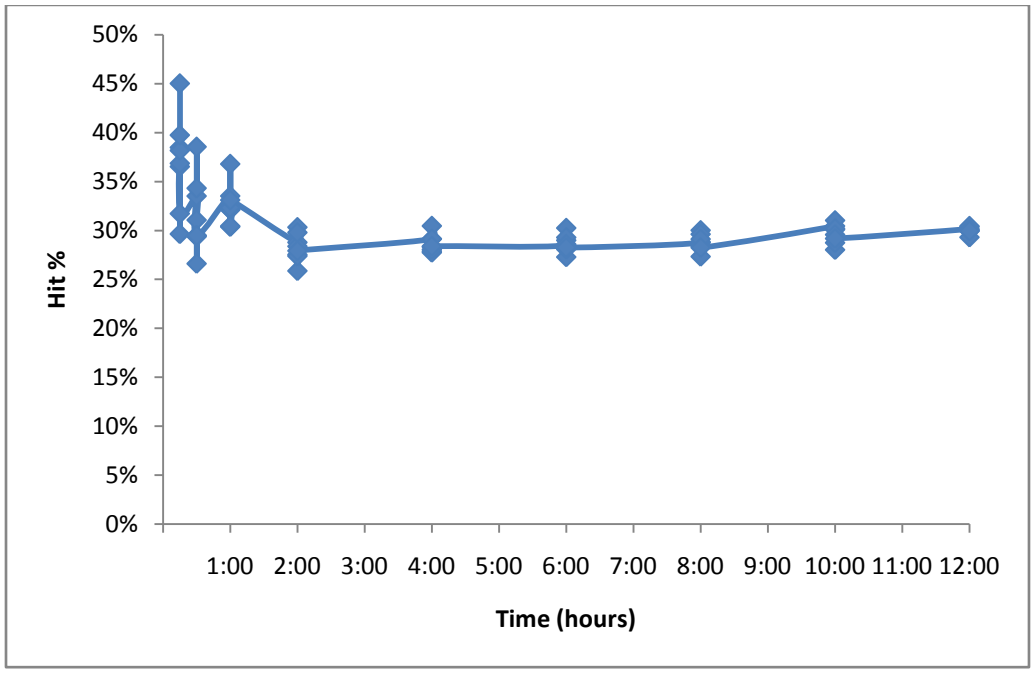

Figure 5.3: Variance in Hit Percentage vs. Test Length

The graph in Figure 5.3 shows the distribution of hit percentages for each test run of varying length. As test length increases, the variance decreases. Based on these results, we chose to use a test length of two hours for our experiments, as this was long enough to eliminate a large portion of the variance but also short enough to allow us to run more experiments within our time constraints.

With these results, we were able to perform actual experiments to determine the effect of latency on actions with varying levels of precision and deadline requirements, regardless of hardware. First, we tested the precision space by varying the sizes of the game's tanks and performing tests with a variety of latency values. Next, we tested the deadline space by varying the tank's bullet speed and running experiments with various latencies. Lastly, we tested the precision and deadline space by varying both tank size and bullet speed and experimenting with various latency values.

All subsequent experiments performed in the individual precision or deadline spaces used the same precision and deadline intervals and latency values. The intervals and latency values are detailed in Figure 5.4.

Precision and Deadline Values	Latency Values
¼ default	0 ms
½ default	100 ms
1 x default	500 ms
2 x default	1000 ms
4 x default	

Figure 5.4: Precision and Deadline Values and Latency Values used in Experiments

The latency values show in Figure 5.4 were chosen due to their significance in most current online games and the different categories of latency we wanted to represent. A latency of 0 milliseconds, categorized as no lag, between the client and server is ideal and no performance degradation should be experienced. A latency of 100 milliseconds, categorized as normal network lag, is the threshold of delay which can be tolerated in First Person Shooter games [2].

Likewise, a latency of 500 ms, categorized as moderate lag, is widely considered the threshold which can be tolerated in a Massively Multiplayer Online Game, but is significantly detrimental to First Person Shooters. Lastly, a latency of 1000ms, categorized as high/severe lag, is generally considered to be the threshold which can be tolerated in a Real Time Strategy Game, but is too high of a value in which to adequately play most online games, especially First Person Shooters. Each latency value provides insight into the implications our results have on many types of online games. In addition to these values, it is important to note that normal Tank Size is 6 meters by 2.8 meters. The default battle field size for all tests is 800 meters by 800 meters. In all cases, the tests were run with eight tanks divided into four teams of two; all of the tanks are equal, and one tank was assigned a latency value between 100 milliseconds and 1000 milliseconds.

5.2: Precision Space

The first set of tests in our experiments involved varying the precision requirements of the game. In order to alter the precision requirements of the game, we modified the size (length and width) of the tanks. As tank size increases, the precision requirements decrease because the size of the target is larger. Conversely, as tank size decreases, the precision requirements increase due to the fact that the target is smaller. Figure 5.4 details the variations of the Tank Size and the latency values which were run for each tank size. Each tank size value was run once with each of the listed latency values.

The results of the precision space experiments are plotted with respect to latency in Figure 5.5 and with respect to tank size in Figure 5.6. The graphs shown in Figure 5.5 and Figure 5.6 show the scores, the kills for the tanks minus the deaths for the tank, of latency induced tanks from two hour experimental tests run with varying Tank Sizes.

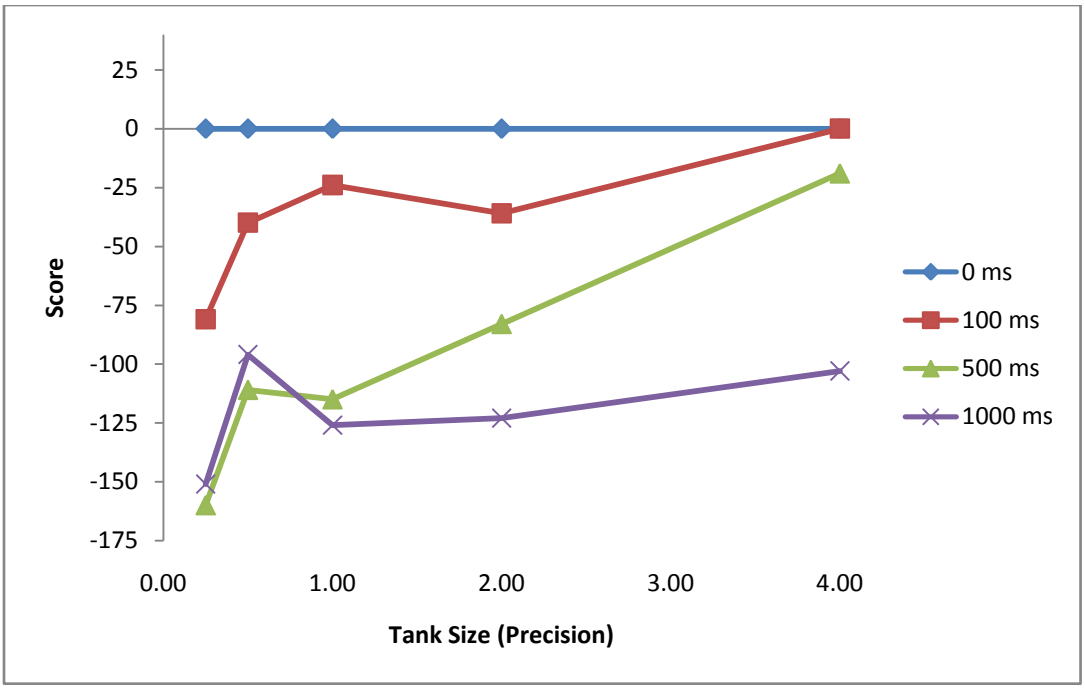

Figure 5.5: Score vs. Tank Size with Respect to Latency

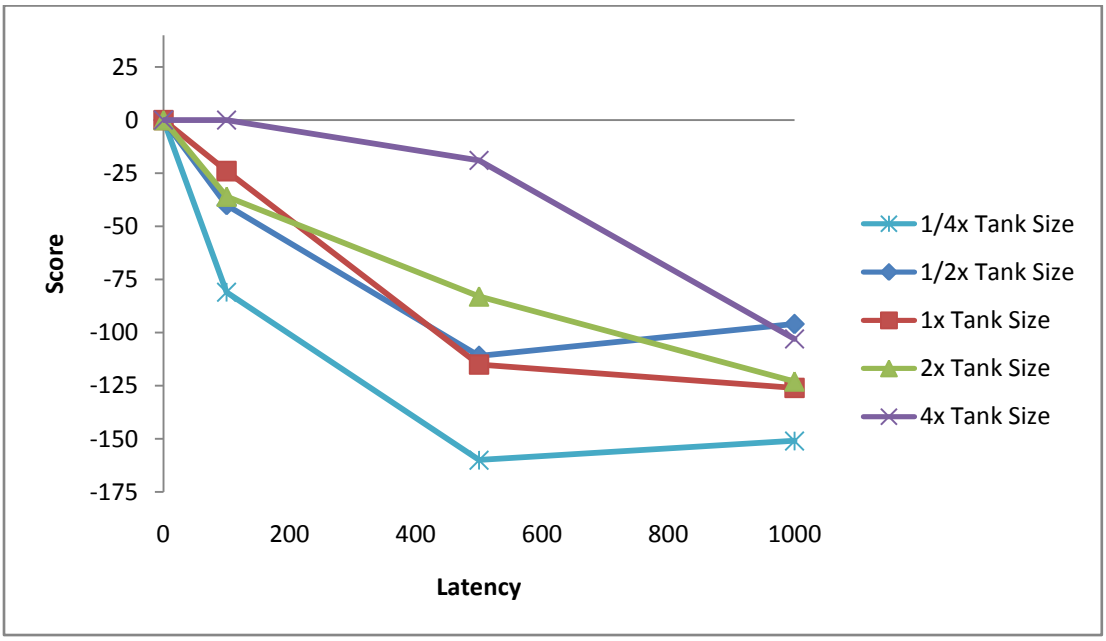

Figure 5.6: Score vs. Latency with Respect to Tank Size

The results show that latency has less of an effect on a tank's score as tank size increases. Conversely, as tank size decreases, the effect of latency on a tank's score becomes

more significant. Also, you can see a significant drop off in performance between tanks with 100 ms and 500 ms of latency, showing that there is a threshold in which latency more significantly impacts performance. The results in the graphs above show that as the precision decreases, the tanks are less adversely affected by latency. Likewise, as the precision increases, the tanks are impacted more severely by the introduction of latency.

The results detailed in Figure 5.5 and Figure 5.6 support our original hypothesis about the precision space. The next section will discuss the results from tests run in the deadline space.

5.3: Deadline Space

The other aspect of gameplay which we examined was deadline. We changed the deadline requirements of the game by modifying the speed of the shots that the tanks fire. Changing the bullet speed has several effects on gameplay in *BZFlag*. Increased bullet speed results in the shots taking less time to reach their target, making it more difficult for opposing tanks to evade these shots. This makes the initial shots from tanks more likely to score hits, making it beneficial to get a shot off quickly. Also, the range at which shots are reasonably likely to hit is increased, meaning the tanks engage each other more quickly, and less time elapses before a tank can effectively rejoin the battle after respawning. Overall, this leads to faster-paced gameplay. For slower bullet speeds, the converse is true; shots are more likely to miss, and tanks must get closer to each other before firing, resulting in a slower overall pace for gameplay.

Because the speed of the bullets has a direct correlation with the speed of gameplay, it is a reasonable representation of movement along the deadline axis in the precision-deadline space. As bullet speed increases, deadline decreases due to the faster pace of gameplay and the need to fire shots as quickly as possible. Conversely, as bullet speed decreases, deadline increases due to the shower pace of gameplay. Since performance is known to degrade more significantly from latency as deadline decreases, we hypothesized that latency would have a greater effect on performance as bullet speed increases.

The default speed for shots in *BZFlag* is 100 m/s, which is four times the default tank speed of 25 m/s. It takes 3.5 seconds for a bullet to travel its default maximum range of 350m, and this is also the reload time during which the tank cannot shoot. The default map is an 800m x 800m square, which means that tanks can shoot about halfway across the map at most, although the chances of hitting are low at maximum range. After the modifications we made to the AI, which are detailed in Section 4.3, the average chance of a tank's shot hitting at normal bullet speeds is about 44%.

In order to properly conduct the bullet speed tests, an adjustment to range was also required. This is because *BZFlag* bases the reload time for the bullets on the amount of time a bullet takes to reach its maximum range. When a bullet reaches maximum range and disappears, the tank may fire again, regardless of changes to the reload time variable. Changing the reload time had a greater effect on gameplay than modifying the range did, so we decided to modify the range and keep reload time fixed at 3.5 seconds. Since the default range is almost half of the map, increasing the range had little effect on gameplay since obstacles generally interfered with shots at distances greater than this. For the slower bullet speeds, the range reduction only eliminated shots that were largely inadvisable (very low chance of hitting).

The results of our experiments with bullet speed are shown in Figures 5.7 and 5.8. The multipliers and latency adjustments are the same as described in Section 5.1.

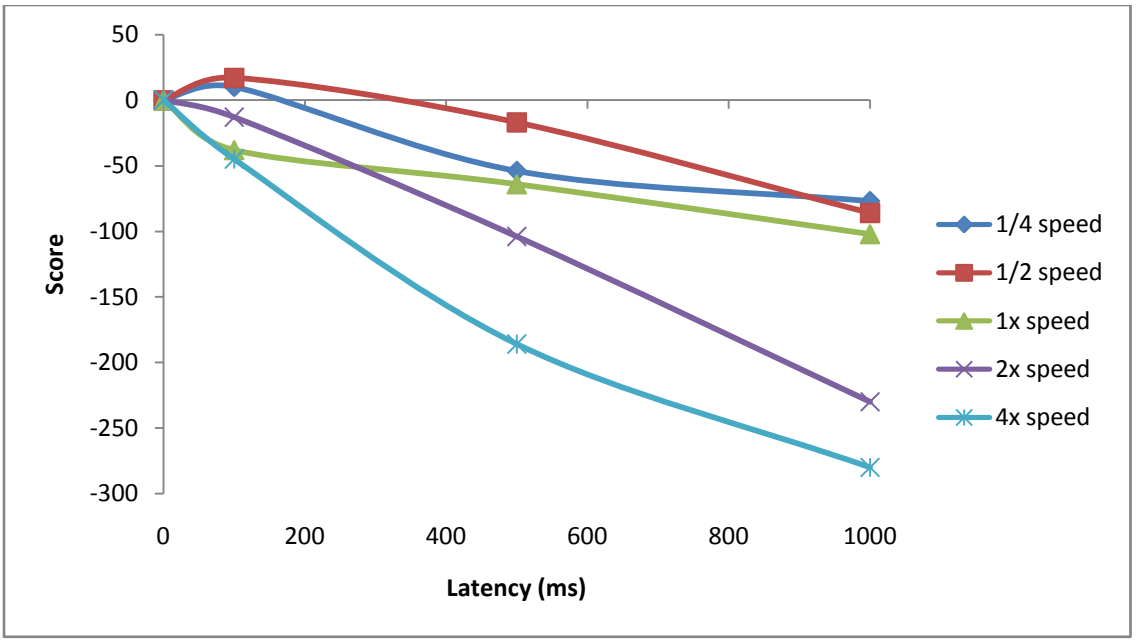

Figure 5.7: Score as Latency Increases

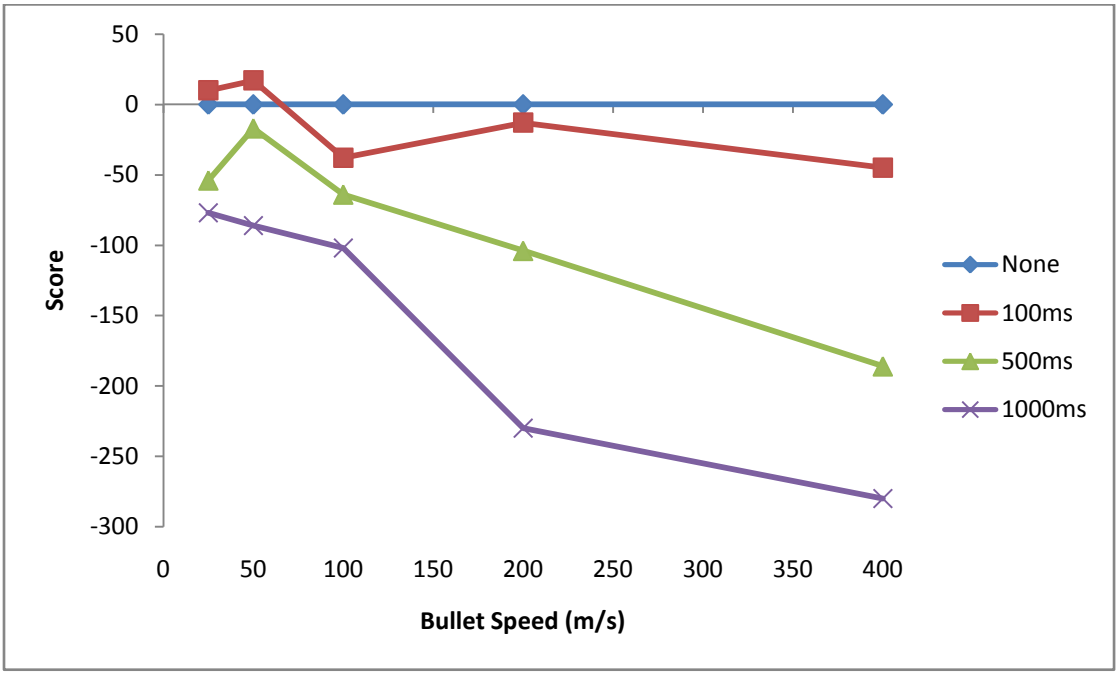

Figure 5.8: Score as Bullet Speed Increases

The two graphs in Figure 5.7 and Figure 5.8 show the performance degradation from latency for the different bullet speed multipliers. This is represented by the scores (kills minus deaths) of the latency-induced tanks. A non-lagged tank was shown to accrue a score of 0 over

time due to its kills and deaths being even. These experiments were run with the settings described In Section 5.1.

Figure 5.7 shows the score trend lines for each bullet speed setting as latency increases. Figure 5.8 shows the same data in the form of trend lines for each latency value as bullet speed increases.

The results show that low latency values have very little effect on a tank's score, regardless of bullet speed. At higher latency values, the speed of the bullets does become a factor; the slower the bullet speed, the higher the latency value needed before a drop-off in performance occurs. Faster bullet speeds result in a greater decrease in performance from latency, and reduced performance can result even from lower latency values.

Again, the results support our hypothesis. Performance decreases steadily as latency increases for all bullet speeds, and the drop-off in performance is sharper with bullet speed than with tank size. This indicates that for *BZFlag*, a modification to the Bullet Speed (deadline) has a greater quantitative effect than the same modification to the Tank Size (precision). The next section will discuss the results when both requirements are modified.

5.4 Precision and Deadline Space

Although our previous results from both the precision space and the deadline space supported our hypotheses, we also decided to run tests modifying both spaces to determine the relationship between them. The tests involved modifying both the tank size and the bullet speed parameters of the game and running tests with no latency and with severe latency.

These tests were focused on the extremes of precision and deadline requirements, and as such only involved four combinations of the limits of the Tank Size and Bullet Speed modifications. The tests were run once with no induced latency and again with one tank receiving 1000 milliseconds of induced latency. The results of the tests are detailed in Figure 5.8.

Bullet Spe		**Tank Size (Precision)**	
		4 x (lowest)	**¼ x (highest)**

	¼ x (lowest)	-71	-51
	4 x (highest)	-161	-201

Figure 5.8: 1000ms Lagged Tank's score vs. Tank Size and Bullet Speed

Figure 5.8 shows the score of the tank with 1000ms of induced latency in the various special case tests. In games with the strictest precision and deadline requirements, the lagged tank has the worst performance when compared with the other special cases. Also, the data shows that in *BZFlag*, deadline requirements influence the performance of the tank more so than the precision requirements. However, while the tank was expected to perform the best relative to the other extreme scenarios when the precision and deadline requirements were the lowest, all of the tanks were guilty of significantly more team kills in this scenario due to the size of the tanks, thus driving down scores. This was due to the fact that the tank AI does not take into account their increased Tank Size when attempting to evade bullets in the game world and as a result, they are hit by shots that they think they have evaded. However, it can still be said that as precision and deadline requirements increase, the effect of latency becomes much more significant.

Through the tests described above, we were able to determine that actions in *BZFlag* are affected differently based on their precision and deadline requirements. We measured the effects of latency on both precision and deadline individually and then the combination of the two. However, although our tests were run using *BZFlag*, the results of our tests are applicable across online games in general. All game actions can be described in terms of their precision and deadline requirements and thus, our results can be applied to those actions to determine how susceptible they are to network latency. The next section will discuss the implications of the results obtained from all of the experiments.

5.5: Implications of Results

BZFlag in general is fairly low on both the precision and deadline scale for a first-person shooter game. Default bullet speed is slow relative to the speed of the tanks, meaning misses are common and even if a tank is hit, it generally has time to shoot back before being killed.

The target to hit is fairly large and there is no height component to the shots, so precision along the z axis is not required when firing. This means that the default settings for *BZFlag* generally do not match those of other first-person shooter games. However, some of the experiments involving faster bullet speed and smaller tank size do approach the precision other shooting games require. If another game has similar values for bullet speed, target size, target speed, and rate of fire, then the data from the appropriate experimental test run can be used to give an approximation for the effect of latency on that game.

One major drawback in applying these results to other games stems from the fact that only a tank's shots were lagged in this simulation. A tank with latency could still evade other tanks' shots in real-time, and could respond instantly to other tanks' movements, shots fired, and deaths. Network latency would affect all these actions as well and not just the shooting component. Therefore, it can reasonably be said that in a game with similar precision and deadline requirements to *BZFlag*, the performance degradation from latency would be greater than the data represented here. The *BZFlag* data could therefore be used to quantify a lower bound for performance degradation, but it could not accurately be used to provide anything more than a rough estimate of the actual effect of latency in another game.

Chapter 6: Conclusion

The goal of our research was to provide definition for the effect of latency on the precision-deadline space by quantifying the effect of latency on actions with varying precision and deadline requirements. Our hypothesis was that actions are affected differently by latency depending on these requirements. The motivation for this research was to provide a model by which developers can categorize all actions within a game and also approximate the effect of latency on those actions.

To do this, we chose to modify a publically available game called *BZFlag* to allow us to induce controlled latency to robot players in the game, then run a series of experiments and analyze the results. While *BZFlag* had some qualities that made it suitable for this purpose, there were design choices in the game that are not standard for online games of its genre, and which created complications when implementing the changes we desired to make.

We decided to run tests in the precision space, the deadline space, and finally in both spaces simultaneously. In order to collect data on the effect of precision modifications, we chose to modify the tank size. Similarly, in order to examine the effect of deadline, we adjusted the bullet speed. We then ran a series of two-hour experimental test runs with varying latencies. Through our experiments, we were able to produce a series of data to define the effect of latency over the precision-deadline space. While the experiments we ran with *BZFlag* supported our hypothesis and provided some quantifiable data about performance degradation due to latency, there are still limitations when applying this data to analyze performance degradation in other games.

While *BZFlag* has characteristics that make it different from many online games, the results obtained from our experiments are still applicable to other games. Our results can be used as an effective lower bound when analyzing the impact of latency on actions in another game that have similar precision-deadline requirements. More generally, our results support the theory that actions requiring high precision or deadline are more adversely affected by latency than actions with lower precision or deadline requirements.

6.1: Future Work

Additional research into the precision-deadline space could involve making further adjustments to the *BZFlag* code to allow for more accurate test results. In particular, implementing code to induce latency to all aspects of a tank's actions, not just shooting would be needed. In order to do this, the collision detection code would need to be migrated to the server-side, and the existing `LatencyManager` functions modified to include reading messages as well as sending them.

Other options for exploring the precision-deadline space might include modifying the *BZFlag* code to take into account the bullet radius during collision detection, something it currently does not do. Changing the size of a bullet would modify the precision requirement and could be used to simulate games where shot accuracy is a lesser consideration due to an explosion splash radius. Finally, tank speed is another option that can be modified in order to simulate the movement speeds of other games.

Finally, another way to further improve these models would be to conduct a user study with *BZFlag*. A user study could provide a model that incorporates player skill into determining the effects of latency. However, this would require a code modification, as the current code only applies the latency to robot players, not human players.

All of these suggestions can be used to further and improve upon the research presented in this report. This is not an exhaustive list by any means, but it provides options which can and should be explored in projects which intend to build upon the research detailed in this project.

Works Cited

[1]Leigh Alexander. World of Warcraft Hits 10 Million Subscribers In: Gamasutra, 22 Jan. 08 Online at: http://www.gamasutra.com/php-bin/news_index.php?story=17062

[2] Mark Claypool and Kajal Claypool. Latency and Player Actions in Online Games, Communications of the ACM, Volume 49, Issue 11, November 2006. Online at: http://www.cs.wpi.edu/~claypool/papers/precision-deadline/

[3] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003, In Proceedings of ACM Network and System Support for Games Workshop (NetGames), Portland, OR, USA, September 2004. Online at: http://www.cs.wpi.edu/~claypool/papers/ut2003/

[4] James Nichols and Mark Claypool. The Effects of Latency on Online Madden NFL Football, In Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Kinsale, County Cork, Ireland, June 16-18, 2004. Online at: http://www.cs.wpi.edu/~claypool/papers/madden/

[5] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The Effect of Latency on User Performance in Warcraft III, In Proceedings of ACM Network and System Support for Games Workshop (NetGames), Redwood City, CA, USA, May 2003. Online at: http://www.cs.wpi.edu/~claypool/papers/war3/

[6] Evaluating dead reckoning variations with a multi-player game simulator In: Network and Operating System Support for Digital Audio and Video (NOSSDAV 2006), pp. 20--25, ACM Press (ISBN: 1-59593-285-2) http://home.ifi.uio.no/paalh/publications/files/nossdav2006-palant.pdf

[7]Gettimeofday function for windows In: C/C Programming Blog, Dec 26 2007. Online at: http://www.openasthra.com/c-tidbits/gettimeofday-function-for-windows/

[8] 4x4 Xbox 350 Ladder In: Global Gaming League. Online at: http://www.ggl.com/index.php?controller=MatchManager&method=view&id=150

[9] *BZFlag* System Requirements In: *BZFlag* Wiki. Online at:
http://my.bzflag.org/w/Download#System_Requirements