Reducing the Negative Effects of Inconsistencies in Networked Games

Cheryl Savery
School of Computing
Queen's University
Kingston, Ontario, Canada
cheryl.savery@queensu.ca

T.C. Nicholas Graham
School of Computing
Queen's University
Kingston, Ontario, Canada
nicholas.graham@queensu.ca

ABSTRACT
Networking is a key component of digital games, with many featuring multiplayer modes and online components. The time required to transmit data over a network can lead to usability problems such as inconsistency between players' views of a virtual world, and race conditions when resolving players' actions. Implementing a good consistency maintenance scheme is therefore critical to gameplay. Sadly, problems with consistency remain a regular occurrence in multiplayer games, causing player game states to diverge. There is little guidance available on how these inconsistencies impact player experience, nor on how best to repair them when they arise. We investigate the effectiveness of different strategies for repairing inconsistencies, and show that the three most important factors affecting the detection of corrections are the player's locus of attention, the smoothness of the correction and the duration of the correction.

Author Keywords
Consistency maintenance; game development; usability

ACM Classification Keywords
H.5.3 Information Interfaces and Presentation: CSCW

General Terms
Human Factors

INTRODUCTION
Networking has become a key technology within digital games, as multiplayer interaction has become an almost ubiquitous feature. The time required to transmit data over a network can lead to usability problems in networked games, such as inconsistency between players' views of a virtual world and race conditions when resolving players' actions. Implementing a good consistency maintenance scheme is therefore critical to game play.

Sadly, problems with consistency remain a regular occurrence in multiplayer games, including some top-selling titles, and these problems can have a huge impact on player experience. For example, when Battlefield 4 [2] was released in November 2013, players protested about the numerous lag issues with the game. Sites such as IGN [12] and Reddit [18] were inundated with negative comments from users. Even today, the game continues to suffer from issues such as rubberbanding, where characters are pulled back to previous positions, and players being killed despite being under cover. While Battlefield 4 is one of the most egregious cases, it is certainly not unique. In a 2009 survey of 389 reviews of networked games, Pinelle et al. found that 49 of the reviews specifically cited gameplay problems related to network latency [17]. For example, in his review of Rainbow Six: Lockdown, William Harms commented that "*there were constant issues with ghosting, where an enemy would simply vanish right before you doubletapped him*" [11].

What is it about game networking that makes it so difficult to provide a consistent view of the game world to all players? The primary culprits are network latency, that is, the time required for updates to travel over the network, and limited bandwidth, meaning there is sometimes more data to be updated than can be transmitted without delays. To overcome these restrictions, games rely on a variety of techniques to maintain consistency, such as *dead reckoning* [16], which uses prediction to estimate the positions of remote players and other game objects between network updates, and *local lag* [13], which delays local updates in order to improve overall consistency. Regardless of the technique used, the fact that it takes time to transmit state updates over a network means that the game state can never be perfectly synchronized. Inconsistencies are thus inevitable.

Despite the large body of research on techniques for maintaining consistency in networked games, the issue of how to *repair* inconsistencies when they are detected has been largely overlooked. Inconsistencies can be repaired immediately, possibly resulting in a confusing or jarring experience such as seen in Battlefield 4. Or gameplay can be adjusted to mask the inconsistency until it can be repaired. There is little guidance available as to which of these broad approaches (and their many variants) best reduces the impact on player experience. To address this, we conducted an experiment that provides answers to three broad questions:

- How noticeable and annoying are corrections in networked games?

CHI PLAY'14, October 19–21, 2014, Toronto, ON, Canada.

http://dx.doi.org/10.1145/2658537.2658539

- How do different gameplay conditions affect a player's reaction to these corrections? and
- What techniques may make corrections less annoying to game players?

Our study tested 18 people in five variations of a custom game that implemented several techniques for error correction. For each correction, we recorded whether or not players noticed the correction, and if they did notice it, the degree to which they found the correction annoying. Some of our results were largely expected, but are confirmed here for the first time: corrections are most problematic when they occur within the player's locus of attention, and corrections are far more annoying when they impact the game situation. Other results were surprising. Smoothing corrections over even a 250 ms interval has an enormous effect on how frequently corrections are noticed. Performing corrections with a gradually changing velocity is surprisingly better than using constant velocity. And the likelihood with which a player sees that a correction has taken place depends more on the correction's velocity than its magnitude.

This paper represents the first study of the impact of error repair on player experience in networked games, and makes three important contributions. First, we provide a list of factors that influence how corrections affect player experience. Second, we evaluate techniques for reducing this impact. Third, we provide specific guidelines for the rate at which inconsistencies should be corrected.

BACKGROUND AND RELATED WORK

In networked games, inconsistencies arise due to a combination of network latency and the use of prediction by the game client to determine the local game state. Because of the real-time performance requirements of many networked games, some inconsistency must typically be tolerated in exchange for faster feedback to the player and quicker resolution of game events.

Inconsistencies can have a negative impact on player performance and experience [3, 15], such as the confusion caused by sudden warps in position, the frustration of shooting directly at another player and missing [4], or the bafflement of suffering damage from a grenade while protected by an invincibility shield [1].

When an inconsistency between the views of two players is identified (e.g., when a prediction algorithm has moved an avatar to the wrong location), the error must be repaired – and since errors often involve a visible object in the game, the repair must be visually understandable. As we describe in more detail below, some techniques provide smoother corrections (e.g., interpolating between the incorrect and correct locations over a time period) but at the cost of prolonging the inconsistency, in that it takes longer for the two players' views to reach the same state.

To get a better understanding of inconsistencies and how they can be repaired, let's consider two players, Alice and Bob. Alice is controlling her in game avatar. She is moving from left to right and then suddenly turns upward. We see this motion in Figure 1a. Bob is playing on another computer, and there is network latency between their two computers. In Figures 1b, c and d, we see three possible representations of what might be shown on Bob's screen. In all cases, Bob sees Alice's avatar moving from left to right, but when Alice's avatar turns, there is a delay before Bob's computer receives a message specifying that Alice's avatar has changed direction. Until this message arrives, Bob's computer continues to predict (incorrectly) that Alice's avatar is moving to the right. When the update arrives at Bob's computer, Bob's game client detects an inconsistency: since Alice's avatar has been predicted to be moving to the right, but was in fact moving up, her avatar is shown in the wrong location on Bob's screen.

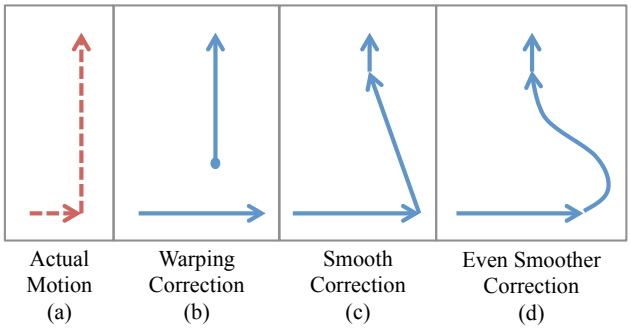

Figure 1. The dashed line (a) shows the motion of a game object that is travelling from left to right and then sharply turns upward. The next three images show the motion of that same object as viewed on a remote display when using (b) warping, (c) simple smooth corrections and (d) smooth corrections with a gradually adjusted correction rate.

This error can be corrected in one of three ways. (1) Alice's avatar can simply be drawn in the updated location. More precisely, the avatar will be drawn in a new predicted position based on the newly received positional update – the prediction is necessary, since the avatar may have moved during the time it took for this update to be transmitted over the network. Bob will see Alice's avatar jump (or "warp") to this new position (Figure 1b). (2) The update can be performed progressively, smoothly moving the avatar to the new location. Bob would see Alice's avatar speed up to join her newly predicted path (Figure 1c). (3) Or, the avatar can be moved with a gradually increasing speed to the new position. Bob would see Alice's avatar change speed more gradually to join its newly predicted path (Figure 1d).

There are significant tradeoffs between these solutions, and to-date, their impact on players' experience has received very little attention. Warping may affect players' immersion in the game world, since a player's attention may be drawn toward any sudden discontinuities in the position or motion of objects in the game [15]. A player may also lose context, for example, if an avatar that was in front of him is suddenly behind him, out of his field of view. Smooth corrections may reduce the jarring effect of corrections [20]; however, they prolong the inconsistency, which may be unacceptable in some game situations. Fiedler [8] suggests the rule of thumb of moving 10% toward the true position during each frame, but if the correction is large, simply warping to the new position.

Figure 2. In the Space Rock-it colouring game, players colour the asteroids by bumping into them with their ship.

More sophisticated forms of smooth correction are possible; e.g. varying priorities may be given to different objects in the game to ensure the objects of higher importance deviate the least from their true position [6]. While these algorithms are promising, and while ad hoc rules of thumb can be useful, there has been to-date no experimental validation of the impact of these techniques on user experience. There has been little effort to characterize when smooth corrections are effective, and to determine when and how they should be applied.

USER STUDY

As indicated previously, our research addresses three broad questions: how noticeable and annoying corrections are in networked games, how different gameplay conditions affect players' reactions to corrections, and what techniques make corrections less annoying for game players.

To address these questions, we conducted a user study using a custom-built game that allowed us to inject errors and corrections into the game and to obtain player feedback. The game – *Space Rock-it*, as shown in Figure 2 – features two space ships and asteroids moving about in a 2-D world. Participants played *Space Rock-it* under five separate gameplay conditions, where they either observed the motion of the objects on the screen or controlled the motion of one of the space ships. Consistency errors were periodically injected into the game, with varying magnitude and varying correction styles. *Space Rock-it* is intentionally simple in order to allow easy isolation of gameplay conditions and to emphasize the importance of corrections on player experience. We now describe how the game allowed us to address our research questions.

Q1: How Noticeable and Annoying are Corrections?

As participants played *Space Rock-it*, errors of differing magnitude were randomly injected into the game and then corrected using a variety of algorithms. Players were instructed to press the space bar when they saw a correction, and then rate how annoying they found the error. This allowed assessment of what magnitude of errors people perceive, under which conditions, and how annoying they found them.

Table 1. Summary of Game Conditions

Condition Name	Task	Game Entities	Ship Corrected
Observing - Single	Observe Only	One	Observed Ship
Observing - Many	Observe Only	Many	Observed Ship
Colouring	Colour Asteroids	Many	Other Ship
Controlling	Colour Asteroids	Many	My Ship
Shooting	Shoot Other Ship	Many	Other Ship

Q2: How Do Gameplay Conditions Affect Reactions to Corrections?

In addition to the question of how annoying people find corrections to be in general, we wished to investigate the impact of three key factors on how players notice and react to corrections. These factors are *distraction*, *locus of attention*, and *involvement with gameplay*. First, the amount of activity and number of objects or clutter in the game interface may *distract* players and affect their ability to notice corrections. Second, at any given time, people focus on a single area of the display, and the proximity of a correction to this *locus of attention* may affect how likely players are to notice it [21]. Finally, the very *act of playing a game* may affect a player's ability to notice corrections: playing a game may either distract players, causing them to not notice the corrections, or it may exacerbate the effect of a correction if it breaks immersion or affects gameplay performance.

To investigate these questions, we created five game conditions within the *Space Rock-it* game (see Table 1). The rationale behind each condition is discussed below.

First, to investigate whether or not a visually busy game might distract players and therefore mask corrections, we used two conditions which differed only in the number of potentially distracting objects on the screen. The task was simple: watching the screen and reporting corrections when they were observed. In the *Observing - Single* condition, the participants observed the motion of a single pink space ship on an otherwise empty display. In the *Observing - Many* condition, the participants again observed the motion of the pink space ship, but there were also a second green ship and 40 asteroids moving about on the screen. In both conditions, corrections were applied to the pink ship at random intervals. No corrections were applied to the green ship or the asteroids. Comparing the number of corrections detected by the participants in these two conditions allowed us to see whether the distractions had any effect.

In the *Observing* conditions described above, the participants were passive observers of the pink space ship, allowing them to fully focus their attention on it. By adding elements of game play to *Space Rock-it*, we manipulated the players' locus of attention. In the "Colouring Game", players control the green ship with the keyboard, and colour asteroids by bumping into them. At the start of the game, all of the asteroids are yellow in colour. If the green ship hits an asteroid, it turns green. If the pink ship hits an asteroid, it turns pink. This Colouring Game was used to create two conditions. In the *Controlling* condition, the corrections all occurred to the green space ship being controlled by the player, where we

presumed the player would be focusing most of her attention. In the *Colouring* condition, the corrections all occurred to the pink, non-player controlled ship where we expected the player would be paying much less attention. The "Shooting Game" variant enable a third condition in which the player needed to pay attention to two locations on the display. In the *Shooting* condition, the player moved and aimed the green ship while attempting to hit the pink ship with missiles. Corrections were applied to the pink target ship. By comparing the number of corrections detected by the participants in the *Controlling*, *Colouring* and *Shooting* conditions, we determined the degree to which the player's locus of attention affected their ability to notice corrections.

Finally, to investigate how playing a game impacted a player's opinion of corrections, we compared the *Observing - Many* and the *Controlling* conditions described above. In both conditions, the player is focused on the ship being corrected. The conditions differ only in that in the *Observing - Many* condition, both ships move on their own, while in the *Controlling* condition, the player attempts to colour asteroids by controlling the green ship that is experiencing the corrections. We could thus determine how the act of playing a game affected player perceptions by comparing the number of corrections detected and the player ratings of the corrections for these two conditions.

Q3: How Do Error Repair Techniques Impact Player Experience?

Our third major question addresses the effect of different repair techniques on player experience. We implemented three repair strategies in *Space Rock-it*: warping, smooth corrections with a constant correction rate, and smooth corrections with a variable correction rate. To provide guidelines for repairing inconsistencies, we compared warping corrections to smooth corrections and we compared smooth corrections with a constant correction rate to those with a gradually adjusted rate. We also calculated detection limits for the various corrections based on the time interval over which the correction was performed.

When performing smooth corrections, the simplest approach is to pick a target location where the game entity should arrive at some point in the future and then adjust the entity's velocity to smoothly move to that location (Figure 1c). The problem with this approach is that, although there are no instantaneous jumps in the position of the entity, there is a sharp and sudden change in its velocity. Smed and Hakonen have suggested that gradually changing the velocity (Figure 1d) would make the correction less noticeable and thus be preferable [20]. We sought to confirm this by implementing two versions of smooth corrections. In the first version, we set a time interval (ranging from 250 ms to 1000 ms, refer to the *Procedure* section for specific values) over which the correction is to be completed and then move the entity toward the correct location at a constant rate. This is shown by the solid blue line in Figure 3. In the second version, we perform the correction over the same time interval, but initially move the entity slowly, increasing the correction rate over time as shown by the dashed red line in Figure 3. The correction

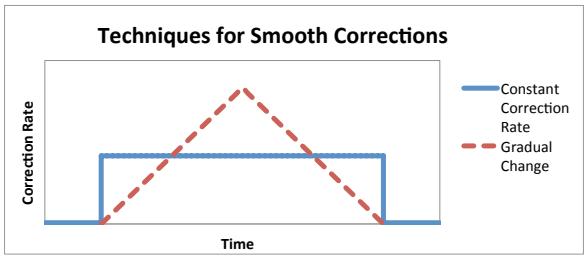

Figure 3. Corrections may be done using either a constant or variable correction rate

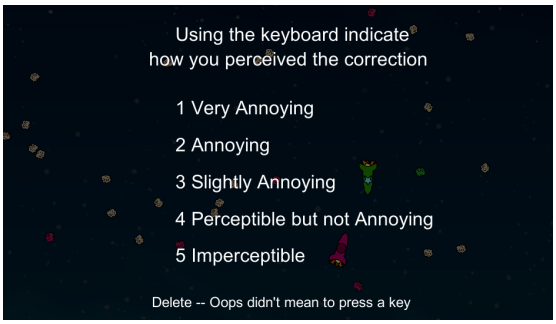

Figure 4. When the participants noticed a correction they rated the degree of annoyance on a scale of 1 to 5

rate increases linearly up to the mid point of the correction and then decreases linearly. In both cases, the correction is completed over in the same amount of time, and the average correction rates are identical. Note, however, that when the correction rate is changed gradually, the correction rate is actually higher in the middle of the correction. We compared the number of corrections detected and the player ratings for these two forms of smooth correction.

We expected that if smooth corrections are performed increasingly slowly, there eventually comes a point where the corrections are undetectable. In order to compute the detection limits, we included a range of correction sizes and correction times in the study. The specific values are listed in the *Procedures* section.

Method

18 participants (16 male and 2 female) were recruited from a local university. All of the participants were regular game players who played fast-paced action games for a minimum of four hours per week. We specifically recruited regular game players, as our earlier work has shown that non-gamers have significantly less ability to detect anomalies in games [19].

Procedure

The participants played the game seated at a computer with a 27 inch monitor and used the keyboard both to control their on screen avatar (a space ship) and to provide feedback when they observed a correction. Although the study investigated repairs to inconsistencies in networked games, we used a single player game in which errors and the subsequent corrections were synthetically generated. This allowed us to control the frequency and magnitude of repairs to inconsistencies.

The study began with a brief training session in which the participants observed the normal motion of the space ship for approximately thirty seconds. The ship then suddenly warped to a new position and the participants were asked to press the space bar when they saw this happen. At this point, the rating system (Figure 4) appeared on the screen and was explained to them. The participants continued to observe the space ship for an additional minute and pressed the space bar each time they noticed a correction.

Following this training period, the participants played the five different game conditions listed in Table 1, with each condition lasting approximately eight minutes. The conditions were always played in the same order:

- *Observing - Single*: Observing one ship
- *Observing - Many*: Observing two ships and asteroids
- *Colouring*: Colouring game with corrections occurring to the other ship
- *Controlling*: Colouring game with corrections occurring to the player controlled ship
- *Shooting*: Shooting game with corrections occurring to the target ship

We did not randomize the order of the conditions, because we wanted the players to become thoroughly familiar with the motion of the space ships and asteroids before beginning the game play conditions. The results indicate that any learning effect had little impact on the results, as we found no significant difference in the total number of corrections detected between the *Observing - Single*, *Observing - Many* and *Controlling* (first, second and fourth) conditions.

During each condition, corrections occurred to the position of one of the two space ships. The players were asked to indicate when they noticed a correction and to specify the degree to which they found the correction to be annoying. When the players noticed a correction, they pressed the space bar. The game then paused and the participant rated how annoying they would have found a similar correction had it occurred in an actual game (Figure 4). The rating scale was based on the *Mean Opinion Score* (MOS) [14], a metric established for subjective quality measurements of audio and video signals. The MOS is based on human perception and has values ranging from 1 to 5 with 1 indicating the poorest quality and 5 indicating no signal degradation. This scale has been adapted and used by researchers to measure video game quality, particularly when affected by network latency [7, 22, 23].

The participants also had the option of pressing the delete key to indicate that they had accidentally pressed the space bar. While the game was paused, the positions of the space ships and the asteroids were dimly visible in the background behind the text. This enabled the players to remain focused on the game while providing feedback about the corrections. The game resumed immediately after the player pressed a number key to rate the correction. The order of the errors was randomized and the errors and corrections were uniformly distributed over three to eight seconds.

Following the study, we conducted a semi-structured interview with each participant to elicit additional information about how they rated the corrections and the types of corrections they found most annoying.

During each condition, there were 60 corrections. We varied the corrections along three axes:

- the magnitude of the correction,
- the correction time – the time over which the error was repaired, and
- the smoothness of the correction

Each of these axes is described in more detail below.

Magnitude of the Correction

The magnitude of a correction is measured in centimetres on the display. The space ship was 4 cm long by 1.5 cm wide and travelled at a speed of 12 cm per second. This size was chosen based on aesthetics and the requirement that the ship be easily visible on the screen. The speed was selected to make the ship comfortable to control. In setting the magnitude of corrections, we considered the range of errors that might typically be found with modern network conditions and based the size of the correction on how far the ship could travel in 125, 250 and 500 ms. With a ship velocity of 12 cm/s, the correction magnitudes were thus 1.5, 3.0 and 6.0 cm respectively.

Duration of the Correction

When repairing inconsistencies, a game developer must choose between making the repair instantly with possibly jarring effects or prolonging the inconsistency and progressively fixing it over time. While it seems intuitive that repairing inconsistencies over time would be superior to warping, this had never been tested prior to this study, and there was no experimental data to indicate how quickly the repairs should actually be made. We performed a pilot study to determine repair times that ranged from very obvious to rarely detected, and based on these, we selected five duration values : 0 ms (warping), 250 ms, 500 ms, 750 ms and 1000 ms.

Smoothness of the Correction

When repairing inconsistencies using smooth corrections, the simplest solution is to adjust the speed of the entity and move it at a constant rate toward the corrected position, as shown in Figure 1c. This has the potential to be jarring for the player as the entity instantaneously changes speed and direction. As shown in shown in Figure 1d, "smoother" corrections gradually adjust the correction rate so that there are no sudden changes in velocity [20]. The actual correction rates for these two techniques are shown in Figure 3. When the correction rate is changed gradually, the actual rate is faster at the midpoint of the correction. In our study, we compared these two options of constant and variable correction rates.

Data Collection and Analysis

The *Space Rock-it* game was used to log data about the corrections and the participant responses. For each correction, we recorded *date and time*, *game condition*, *correction type* (constant or variable correction rate), *magnitude of correc-*

Table 2. Summary of Participant Responses

	Description	Number Reported	False Positives
1	Very Annoying	579	1
2	Annoying	660	5
3	Slightly Annoying	760	2
4	Perceptible but not Annoying	694	19
5	Imperceptible	3	0
	Not detected	2,704	
	Total	5,400	27

tion, *correction duration*, *positions of the ships*, *participant response* (or non-response if they did not detect the correction). We also recorded when participants incorrectly reported corrections when none had occurred.

We analyzed the data using one-way ANOVA to test for effects on the number of corrections observed. Player opinion of the corrections was analyzed with Wilcoxon tests on medians.

RESULTS

The results of our study allow us to address the three broad questions posed earlier in the paper: how noticeable and annoying do players find corrections; how do gameplay conditions affect reactions to corrections; and how do error repair techniques impact player experience?

During the study, participants were presented with a total of 5,400 corrections across all conditions. Participants noticed 2,696, or 50%, of these corrections. Table 2 summarizes the level of annoyance that participants assigned to each correction. There were a large number of reports at each level, indicating that participants used all available categories.[1]

We also collected data to measure the number of times a participant pressed the space bar when no correction had occurred (false positives). There were 27 false positives, representing 0.5% of the total corrections. We believe this to be a sufficiently small value as to not impact the overall results of the study and thus the false positives were simply discarded in the subsequent data analysis.

Q1: How Noticeable and Annoying are Corrections?

Participants noticed 50% of the corrections that occurred during gameplay, and rated 46% of the corrections they noticed as either "Annoying" or "Very Annoying". This is consistent with the anecdotal experience of games such as Battlefield 4 and Rainbow Six: Lockdown, where consistency issues were reported by players and reviewers as significantly negatively impacting player experience. This indicates that inconsistencies in video games are a real problem, and that corrections can be both noticeable and annoying for game players.

[1]Surprisingly, one participant rated three corrections as "Imperceptible", raising the question of why he flagged them at all. During the post-experiment interview, he explained that he had used the "Imperceptible" rating in cases where he was not sure whether he had observed a correction.

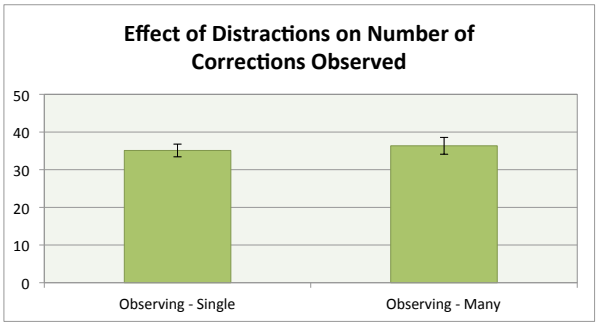

Figure 5. The addition of distractions did not show a significant effect on the number of corrections observed. The error bars shown represent the standard error for the number of corrections detected.

Q2: How Do Gameplay Conditions Affect Reactions to Corrections?

While the data from Table 2 shows that corrections are frequently noticed and are frequently considered annoying, we are primarily interested in what specific factors lead to corrections being more or less visible or annoying. We investigated the impact of three key factors on how players notice and react to corrections. These factors are *distraction*, *locus of attention*, and *involvement with gameplay*.

Distractions

We used the two "observing" conditions to evaluate how placing additional elements in the scene affects players' ability to notice when corrections occur. In both conditions, the players were merely observing the motion occurring in the scene. The first condition, *Observing - Single*, displayed a single pink space ship travelling around the screen, while the second condition, *Observing - Many*, showed two mobile space ships and forty asteroids. In both conditions, the participants were asked to focus their attention on the pink ship and all the corrections occurred to the position of the pink ship.

In each condition, a total of 60 corrections were displayed. In *Observing - Single*, the participants noticed on average 35 of the corrections, compared to 36 in the *Observing - Many* condition (see Figure 5). A paired-samples *t*-test failed to show a significant effect on number of corrections detected from adding the distraction of the second space ship and asteroids ($t(17)=0.98$, $p=0.342$).

Because we did not find a significant difference between these two conditions, for simplicity, we will present only the *Observing - Many* condition in all further discussion of the results, and we will refer to this condition as *Observing*.

Locus of Attention

In a game, a player's "locus of attention" is the point on the screen where the player's visual attention is focused. People direct their focus on a small portion of the visual world and their attention must be redirected to enable them to process information in another region [21]. In both the *Observing* and *Controlling* conditions, we directed participants to focus their attention on the ship to which the corrections were being applied. However, in the *Shooting* condition, the player was required to divide her attention between the ship she was

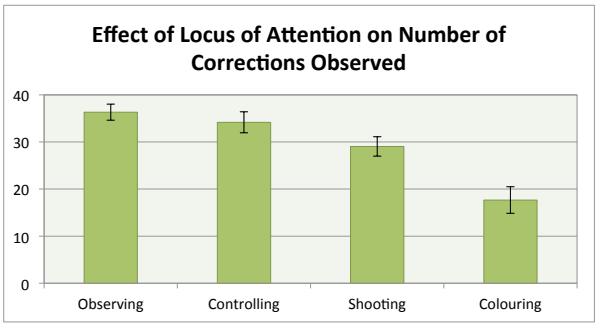

Figure 6. The participants noticed more corrections when their attention was focused on the game entity to which the correction was applied.

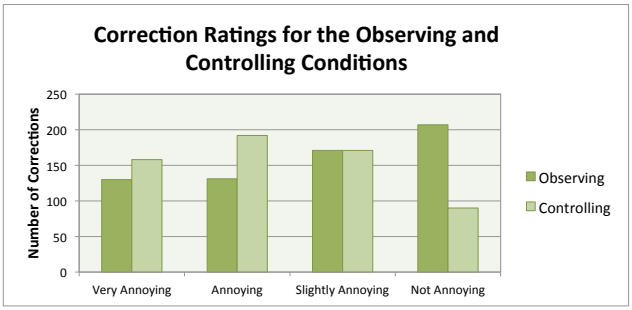

Figure 7. The participants rated the corrections as significantly more annoying when they affected the game play.

shooting at and the ship she was controlling. In the *Colouring* condition, the corrections all occurred to the AI-controlled ship, and the player needed to focus little attention on the motion of that ship. These four conditions therefore represented (1) locus of attention on the avatar being corrected (*Observing* and *Controlling*); (2) locus of attention split between the avatar being corrected and another avatar (*Shooting*); and (3) locus of attention on an avatar other than the one being corrected (*Colouring*).

The number of corrections detected in each of these conditions as well as the standard error is shown in Figure 6. One-way ANOVA showed a significant effect of the locus of attention on the number of corrections observed ($F(3,51)=48.17$ $p<0.001$). Follow-up pairwise t-tests using Bonferroni correction indicated that the number of corrections observed in the *Colouring* condition was significantly less than all the other conditions ($p<0.001$) and that the *Shooting* condition was significantly less than both the *Observing* ($p=0.001$) and the *Controlling* ($p=0.007$) conditions. No significant difference was found in the number of corrections observed between the *Observing* and *Controlling* conditions ($p=0.190$). The participants therefore saw most corrections in the cases where they were focused on the avatar being corrected, less in the split-attention case, and less yet when attention was focused on another avatar.

To investigate more deeply peoples' ability to perceive corrections while focused on another avatar, we performed further analysis on the *Colouring* condition, where players control a green ship that is not being corrected, while corrections are applied to a pink ship. We hypothesized that players would be more likely to notice corrections when the green ship that they control is close to the pink ship that is being corrected. We tested for a correlation between the distance between the two ships and the likelihood of observing corrections. We coded each correction as a 1 if it was detected and a 0 if it was not detected and correlated versus distance between the avatars. We found a weak negative correlation between distance and observation of corrections, ($r(1078)= -0.134$, $p<0.01$), establishing that indeed players were somewhat more likely to observe a correction when it occurred near to their locus of attention.

Involvement with Gameplay

We used the *Observing* and *Controlling* conditions to investigate whether corrections are more annoying when they impact gameplay. In the *Observing* condition, players simply watched the ship, while in the *Controlling* condition, they used the ship to colour asteroids. In both conditions, the player's attention was focused directly on the ship to which the corrections were being applied, and so the only difference between the conditions was whether the observation was passive, or whether the participant was engaged in the game-like colouring activity. In the *Controlling* condition, when a correction occurred, the player's ship moved location, often interfering with the player's intended movement. We hypothesized that this would cause players to rate the corrections as more annoying than in the *Observing* condition.

As described in the previous section, we found no significant difference between the *number of corrections* detected by the participants in each of these conditions. To test whether the corrections were more annoying when they impacted gameplay, we analysed players' ratings of the corrections using the Mean Opinion Score (Figure 4). A Wilcoxon signed rank test indicated that the corrections were rated as significantly more annoying in the *Controlling* condition than in the *Observing* condition ($z = -4.84$, $p<0.001$). Figure 7 shows the correction ratings for the *Observing* and *Controlling* conditions. More corrections were rated as "Annoying" or "Very Annoying" in the *Controlling* condition and more corrections were rated as "Not Annoying" in the *Observing* condition.

Q3: How Do Error Repair Techniques Impact Player Experience?

We now compare the three principle correction techniques of warping, smooth corrections with a constant correction rate and smooth corrections with a variable correction rate and calculate detection limits for the corrections based on the time interval over which the correction was performed. We begin by comparing warping to smooth corrections.

Smooth Corrections Versus Warping

Smooth corrections were performed over four time periods ranging from 250 ms to 1000 ms. We first consider the shortest repair period for smooth corrections, 250 ms. As shown in Figure 8, in all the conditions, fewer corrections were detected when they were performed smoothly over 250 ms than

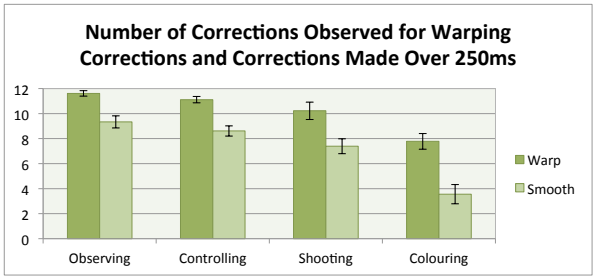

Figure 8. In all conditions, more corrections were noticed when warping was used

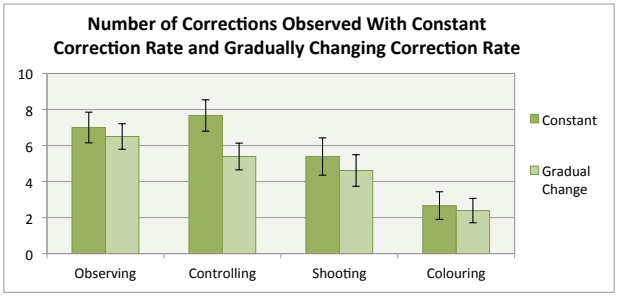

Figure 9. In all conditions, fewer corrections were noticed when the correction rate was adjusted smoothly

if warping was used. In all cases, the difference was significant ($p < 0.001$).

In addition to comparing the number of corrections detected, we also compared the participants' ratings of the corrections. For any correction the participants did not detect, we assigned a rating of 5, "Imperceptible". For each condition, we performed a Wilcoxon test to evaluate whether there was a difference between warping corrections and the corrections performed over a 250 ms window. In each case, the results indicated a significant difference: *Observing* ($z = -11.61$, $p < 0.001$), *Controlling* ($z = -9.72$, $p < 0.001$), *Shooting* ($z = -9.92$, $p < 0.001$) and *Colouring* ($z = -8.30$, $p < 0.001$). This indicates that players found smooth corrections to be less annoying than warping in all five game conditions, when smooth corrections were made over a 250 ms window.

Similar comparisons over longer correction windows (500 ms to 1,000 ms) yielded the same results, that in all cases, smooth corrections were preferred to warping, at the $\alpha < 0.05$ level.

Constant Versus Variable Correction Rate

To investigate the effect of the smoothness of corrections, we first looked at the number of corrections detected by the participants when the corrections were performed using a constant correction rate and when they were performed by gradually adjusting the correction rate (Figure 9). For each condition, there were a maximum of twelve corrections of each type that might have been detected by the participants: three correction sizes (1.5, 3.0 and 6.0 cm) and four durations (250 ms, 500 ms, 750 ms and 1,000 ms). Paired-samples *t* tests showed that the difference was significant for the *Controlling* condition ($t(17) = 5.25$, $p < 0.001$), with fewer corrections observed with variable correction rate. For the other conditions, the results were as follows: *Observing* ($t(17) = 1.23$, $p = 0.235$); *Shooting* ($t(17) = 1.44$, $p = 0.168$); and *Colouring* ($t(17) = 0.74$, $p = 0.472$). When we considered the different correction sizes, there was a significant difference between the number of corrections detected for the small (1.5 cm) and medium (3.0 cm) corrections for both the *Controlling* and *Shooting* conditions. For the large (6.0 cm) corrections, there was no significant difference between the number of corrections detected, likely because these larger corrections were easier to detect and the participants noticed in excess of 60% of the corrections in all of the conditions except *Colouring*.

We consulted the Mean Opinion Score ratings assigned under each condition to determine whether participants found a difference in annoyance of corrections using the constant versus variable algorithms. Wilcoxon tests indicated that there was a significant difference in all conditions except *Colouring*, indicating that variable rate corrections were less annoying in three of four conditions. The results were as follows: *Observing* ($z = -2.47$, $p = 0.014$), *Controlling* ($z = -6.69$, $p < 0.001$), *Shooting* ($z = -2.11$, $p = 0.035$) and *Colouring* ($z = -0.38$, $p = 0.702$).

A review of the number of positive and negative ranks reported by the Wilcoxon test for each correction size and correction duration were similar, indicating that this difference is applicable over the entire range of correction sizes and durations tested in our study.

Correction Detection Limits

We calculated the detection limits for each correction size and test condition. We use a standard psychophysical definition of detection limit as the level at which 50% of the participants notice the stimulus [9]. The detection limits were measured in milliseconds. That is, we sought the minimum correction duration in milliseconds where 50% of the participants detected the correction. In all cases, we noticed that the detection limit for the large (6 cm) corrections was approximately twice as large as for the medium size (3 cm) corrections. Similarly the detection limit for the medium size corrections was approximately twice as large as for the smallest (1.5 cm) corrections. For this reason, we report the detection limits in terms of the correction rate (cm/s) instead of in terms of the time over which the correction was performed and combine the detection limits for the three correction magnitudes. A correlation of the detection limits in cm/s versus the size of the correction validated this approach, showing no significant correlation for any of the conditions: *Observing* $p = 0.080$, *Controlling* $p = 0.575$, *Shooting* $p = 0.812$ and *Colouring* $p = 0.251$.

In Figure 10, we show the average detection limit for each condition. The *Observing* condition had the lowest detection limit ($M = 5.0$, $SE = 0.4$), followed by the *Controlling* condition ($M = 6.4$, $SE = 0.8$) and then the *Shooting* condition ($M = 7.5$, $SE = 0.7$). The *Colouring* condition had a much higher detection limit ($M = 26.2$, $SE = 4.1$). For the first three conditions, the detection limit is close to half the normal speed of the space ship in the game. In the *Colouring* condition, where the corrections occurred to the AI controlled ship and the player was paying less attention to that ship, the limit was much higher at over twice the normal ship speed.

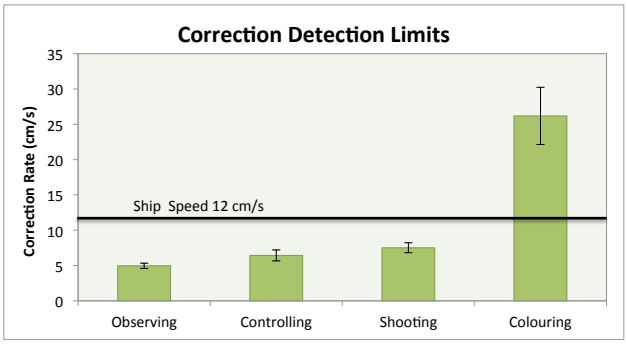

Figure 10. For the *Observing*, *Controlling* and *Shooting* conditions, the detection limit arose at a correction rate of approximately half the normal ship velocity.

SUMMARY OF RESULTS

We found that the primary factor significantly affecting the detection of corrections was the player's locus of attention. Distractions that did not shift the player's focus had little impact on the number of corrections detected.

We found that for the corrections in our study, smooth corrections were vastly superior to warping. Also, based on the player ratings of the corrections, we found that it was significantly better to make gradual adjustments to the correction rate rather than performing the correction at a constant rate.

Finally, we found that the ability of players to detect corrections is based on the speed of the correction, not its magnitude. When an entity is within the player's locus of attention, its speed can be adjusted by approximately 50% relative to its normal speed. Significantly higher adjustments can be made when the entity is outside the locus of attention.

DISCUSSION

The first objective of our study was to determine how noticeable and annoying corrections are. We found that corrections are a real problem in video games and players rated a significant proportion of them as either "Annoying" or "Very Annoying". All of the participants rated at least some of the corrections as "Very Annoying".

Our next objective was to determine which factors affected players' abilities to notice corrections and the degree to which they found the corrections annoying.

We were surprised to find no significant difference in the number of corrections detected as a result of adding distractions to the scene in our two observing conditions. In the first condition, the scene showed only a single space ship and in the other, a second ship and 40 asteroids had been added to the scene. In both conditions, the corrections all occurred to the same pink ship on which the participants were instructed to focus their attention. The additional objects all moved around the scene at a constant speed and did little to attract the participants' attention. Grad et al. [10] have shown that users tend not to notice peripheral information unless it is flashing or animated. Thus, we believe it is likely that the objects we added to the scene did not cause the participants to shift their focus away from the pink ship. It is quite possible that, if the motion of the other ship and asteroids was discontinuous or included some other unexpected motion, the distractions might have had an effect. This tells us that, in general, we cannot rely on distractions to mask the effects of repairs to inconsistencies.

Interestingly, during the post-hoc interviews, 11 of the 18 participants felt that the addition of the second space ship and the asteroids did distract them. It appears, however, that most of the participants thought that they were just momentary distracted; for example, participant 15 stated "*Once I got distracted 'cause I was looking at the green ship.... Then, I thought wait – I shouldn't be watching that.*" and participant 3 commented "*I would start wandering off and think let's see what that green ship is doing, but I am pretty sure I didn't miss anything.*"

The most significant factor we found in determining a player's ability to notice corrections was their locus of attention. Each of the three game-like conditions represented a different level of focus relative to the game object experiencing the corrections. In the *Controlling* condition, the players were controlling the space ship being corrected and their attention was almost completely focused on the point where the corrections occurred. In the *Shooting* condition, the corrections were applied to the target ship and the players were required to divide their attention between the target ship and the ship they were controlling. Finally, in the *Colouring* condition, the corrections were applied to the other ship, yet the position of that ship had little impact on the player. Thus, the players focused little attention on these corrections. Additionally, as the player's attention is shifted away from the point of the correction, fewer corrections were detected (although this distance correlation was weak). Together, these observations tell us that when developing games we need to pay close attention to where the player is looking and attempt to minimize the number and the visibility of corrections in that area.

In addition to knowing which corrections are most easily detected by players, it is also important to know what makes the corrections upsetting or disturbing. We found that corrections were more problematic when they interfered with the player's progress in the game. In the *Observing* condition, corrections were rated as significantly less annoying than those same corrections when they occurred in the *Controlling* condition. Warping corrections were also found to be significantly more annoying than smooth corrections, even when the smooth corrections were completed over a time window as small as 250 ms. In the post-hoc interviews, when the participants were asked which corrections they had found most annoying, they all described either the warping corrections or specific game situations that had affected them. Ten of the participants specifically mentioned the warping corrections; for example, participant 13 commented, "*The leaping one really bothered me a lot.*", and participant 9 stated "*The most annoying ones were when it flashed from one point on the screen to another.*" Eleven of the participants referred to specific gameplay situations that they found annoying; for example, participant 15 stated "*When I was shooting at the pink*

ship and tailing it, it glitched behind me" and participant 2 mentioned "*The most annoying would be when you were trying to hit a ship or an asteroid and it just flies you somewhere totally different.*" Although the players in our study found the smooth corrections less annoying than the warping corrections, we note that corrections are not the only factor affecting player experience. For example, if the correction prolonged the duration of an inconsistency, it is possible that decisions such as who was shot might not be consistent with the players view of the world. Further research is required to study the interactions effects between these game critical decisions and error correction techniques.

In addition to minimizing corrections near the player's locus of attention, developers should attempt to eliminate warping corrections, and should reduce the impact of corrections that affect gameplay. For game developers, eliminating warping corrections can usually be accomplished by replacing them with smooth corrections. Minimizing corrections that might affect game play can be more challenging. One technique that can help is adaptive dead reckoning [5] in which the frequency of update messages for different game objects could be varied based on factors like where the player is focusing and whether or not the object is likely to be affected by game play. The use of remote lag [4] can almost eliminate the need for corrections to the positions of remote game entities, but adds the complexity of making game-critical decisions when players have diverging views of the game world [19].

When corrections are unavoidable, they should be performed as slowly as possible using a gradual change in velocity over time. We found that a correction rate equal to approximately half of the normal ship velocity resulted in only half of the participants detecting the correction. Further studies using different types of motion and different velocities would be required to determine the general applicability of such a guideline.

CONCLUSIONS AND FUTURE WORK

Problems with consistency are a regular occurrence in multiplayer games and these problems can have a huge impact on player experience. However, there has been until now little guidance available on how these inconsistencies impact player experience or how to repair inconsistencies once they are detected.

We have provided insight into how players perceive corrections in networked games and into techniques that can help to minimize the negative effects of corrections. We have performed a user study confirming that smooth correction techniques are significantly better than warping and that smooth corrections with a gradually changing correction rate are preferable over corrections with a constant correction rate. We found that players were most disturbed by warping corrections and corrections that directly affected them in the game, either moving them away from an asteroid they were about to hit, or spoiling their aim when shooting at another other ship. Finally, as a general guideline, we suggest that the speed of corrections should be based on the typical speed of the entity being corrected. We found that the detection limit for corrections was when the entity speed changed by 50%.

In future work, we plan to carry out additional studies, exploring other game situations, other types of avatar motion and other possible ways to distract players from the negative effects of corrections.

ACKNOWLEDGMENTS

We gratefully acknowledge the funding of the GRAND Network of Centres of Excellence.

REFERENCES

1. Aldridge, D. I shot you first! Gameplay networking in Halo: Reach. In *Game Developers Conference* (2011).
2. Battlefield 4. **http://www.battlefield.com/battlefield-4**.
3. Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., and Claypool, M. The effects of loss and latency on user performance in Unreal Tournament 2003. In *NetGames*, ACM (2004), 144–151.
4. Bernier, Y. W. Latency compensating methods in client/server in-game protocol design and optimization. In *GDC* (2001).
5. Cai, W., Lee, F., and Chen, L. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *PADS '99*, IEEE (1999), 82–89.
6. Chandler, A., and Finney, J. On the effects of loose causal consistency in mobile multiplayer games. In *NetGames*, ACM (2005), 1–11.
7. Dick, M., Wellnitz, O., and Wolf, L. Analysis of factors affecting players' performance and perception in multiplayer games. In *NetGames*, ACM (2005), 1–7.
8. Fiedler, G. Networked physics. **gafferongame.com/game-physics/networked-physics**, 2006.
9. Gescheider, G. A. *Psychophysics: the fundamentals*. Psychology Press, 2013.
10. Grad, K., Graham, T.C.N., and Stewart, J. Effective use of the periphery in game displays. In *Future Play*, ACM (2007), 69–76.
11. Harms, W. Review of Rainbow Six: Lockdown. **http://pc.gamespy.com/pc/rainbow-six-4-tentative-title/690399p2.html**, 2006. Accessed: 1-Apr-2014.
12. IGN. **http://ca.ign.com/articles/games/battlefield-4/xbox-one-161397**. Accessed: 9-Apr-2014.
13. Mauve, M., Vogel, J., Hilt, V., and Effelsberg, W. Local-lag and timewarp: Providing consistency in replicated continuous interactive media. *IEEE Transactions on Multimedia 6*, 1 (2004), 47–57.
14. ITU-T Recommendation P.800. Methods for Subjective Determination of Transmission Quality, 1996".
15. Murphy, C. Believable dead reckoning for networked games. In *Game Engine Gems, Volume 2*, E. Lengyel, Ed. CRC Press, 2011.
16. Pantel, L., and Wolf, L. C. On the suitability of dead reckoning schemes for games. In *NetGames*, ACM (2002), 79–84.
17. Pinelle, D., Wong, N., Stach, T., and Gutwin, C. Usability heuristics for networked multiplayer games. In *GROUP'09*, ACM (2009), 169–178.
18. Reddit. **http://www.reddit.com/r/battlefield_4**. Accessed: 9-Apr-2014.
19. Savery, C., Graham, T.C.N., Gutwin, C., and Brown, M. The effects of consistency maintenance methods on player experience and performance in networked games. In *CSCW*, ACM (2014), 1344–1355.
20. Smed, J., and Hakonen, H. *Algorithms and Networking for Computer Games*. Wiley, 2006. ISBN: 9780470018125.
21. Treisman, A. M., and Gelade, G. A feature-integration theory of attention. *Cognitive psychology 12*, 1 (1980), 97–136.
22. Wattimena, A., Kooij, R. E., Van Vugt, J., and Ahmed, O. Predicting the perceived quality of a first person shooter: the Quake IV G-model. In *NetGames*, ACM (2006), 42.
23. Zander, S., and Armitage, G. Empirically measuring the QoS sensitivity of interactive online game players. In *Proc. ATNAC* (2004), 511–518.