Consistency requirements in Multiplayer Online Games

Wladimir Palant
University of Oslo
palant@ifi.uio.no

Carsten Griwodz
University of Oslo
griff@ifi.uio.no

Pål Halvorsen
University of Oslo
paalh@ifi.uio.no

ABSTRACT

Multiplayer online games are becoming increasingly popular as broadband internet connections replace the old modems. However, while available bandwidth grows steadily according to Moore's law, the latency of the internet connections remains almost constant, making it difficult to maintain a consistent game state over a large number of clients that have to be synchronized with each other and with the server(s). This paper introduces a possible solution to the problem by defining the necessary level of consistency through user's perception of the game. While the resulting set of requirements is somewhat difficult to formalize, it is not too restrictive and leaves many options open, some of which are discussed here. Ideally a game where all the requirements are met will appear like a local game to the user.

1. INTRODUCTION

Many games offer you computer players you can compete against. However, while beating computer players is usually challenging, it easily becomes boring because the reactions of your opponent are too predictable. Here lies the key to popularity of multiplayer online games which typically bring many users together in a shared virtual world. Still, bringing people together alone is not enough. One important requirement to bind people to the game successfully is to provide an illusion that other players are next to you regardless their distance in the real world [1].

The connection latency on the internet does not make it easy to create this illusion. To exemplify, we invented a simple game in which we hoped to catch the essence of a first-person shooting game and amplify all the issues caused by latency. The first-person shooting genre was chosen because these games rely heavily on fast action and thus they have the highest requirements when it comes to hiding latency [2].

With this example game we explain a set of user-centered consistency requirements. Other than previous approaches to consistency in games we want to focus on the user's game experience and make it comparable to offline games. In this context consistency means the lack of network artifacts, i.e., the user can not distinguish between a local (lag free) game and a game played over internet.

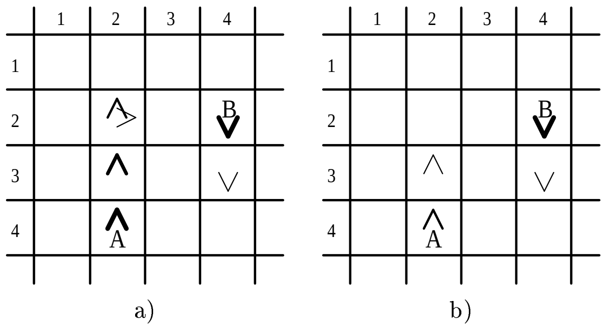

Figure 1: Game scenario example: a) lag free view, b) view owned by B

1.1 Relevant terminology

Local view: the game state as it is available to one particular game client, containing a subset of game objects with their parameters like position and shape as well as general information like game score – all of which might be incorrect.

View owner: the player a particular local view belongs to. Typically every player has his own view.

Lag free view: a view for a theoretical client where all objects are local. This is one possible definition for a "global" view.

1.2 Example game

Our game is played on a board that is shown in Figure 1. We assume a board of infinite size, but action will typically happen in a small area of this board since all players are more interested in winning than in avoiding each other. The players can make one move per second which is either moving one field in their current viewing direction or turning 90° in any direction. To "shoot" at another player, it is enough to look into his direction, if somebody manages to do this his opponent is dead immediately. Though there is a "viewing direction", we assume that the players know everything happening on the board, i.e., there is no "dead angle".

In the game shown in Figure 1a), player A starts in field $(2, 4)$ looking upwards, makes two moves forward and turns

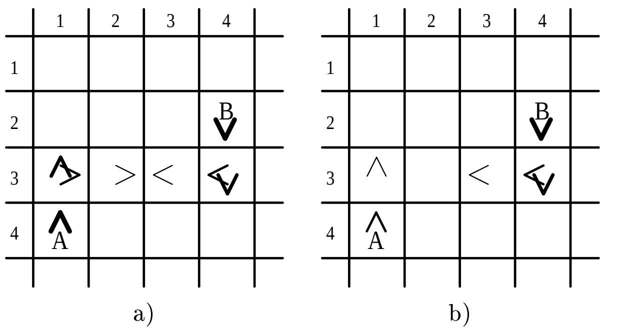

Figure 2: "Dead Man Shooting" situation: a) lag free view, b) view owned by B

to face (and thus kill) player B. During all these maneuvers player B sits still until the last moment where he decides to move one field forward and avoid being killed. In the scheme the different arrows represent the positions and orientations of the players during the game while the thickness of the lines indicates the time at which the event happened (the thicker arrows stand for earlier positions than the thinner ones). We put a letter at the first shown position of a player to indicate which player it belongs to.

1.3 Effects of latency

This makes a pretty boring board game of course, the players can always being hit. However, introducing latency shows a number of interesting effects. If we assume that notifications about players' moves always arrive 2 seconds delayed at the other players and then check what the same game looks like for player B, we get Figure 1b) which gives us an entirely different impression of the same game. Instead of playing "sitting duck" and only avoiding a hit in the last moment, the move from player B can now be interpreted as an attack on A – here B only has to turn to kill A. Too bad that according to the lag free view in Figure 1a), A has already left the position where he is supposed to die.

As we see, due to latency players can get an entirely different view of events happening in the game. We will often see situations where players disagree on whether somebody is currently dead or alive, an extreme example being the "Dead Man Shooting" syndrome (Figure 2). If we observe the game through the lag free view, we notice that both players turn to shoot at each other at exactly the same moment so that they both die. However, in the view of player B that receives all moves from player A with 2 seconds delay this looks differently – B manages to kill A before A gets a chance to do the same. One second later he should receive the message that A shot him as well – a player who he thinks already dead. He has been shot by a "dead man".

1.4 Use of prediction

Current games typically use prediction to alleviate the effects of latency. Predicting the current position of a remote object has the advantage that the predicted position is typically more accurate than the outdated position from the last position update. Furthermore it allows game clients to display a smooth movement trajectory for the object between position updates so that the frequency of position updates can be reduced.

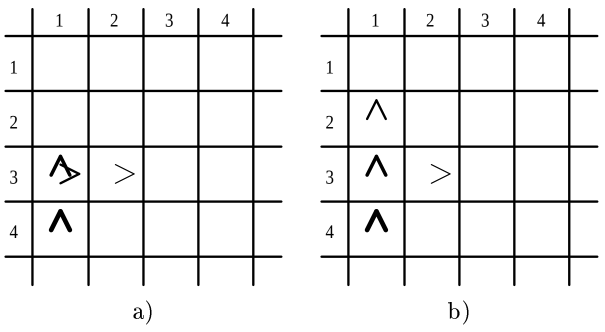

Figure 3: Using prediction: a) the real movement trajectory, b) predicted movement

However, the main drawback is that accuracy of prediction results cannot be relied on. If the game uses prediction and receives a position update that is totally different from the expected (predicted) value, it will need to correct the situation somehow. Usually, the displayed position of the object is changed immediately to the one that is considered correct at the moment.

In our game, there is only one meaningful way to predict positions – assume that the player will move forward. It is very unlikely that a player should stand still in this type of game, and since the board is infinite, we have very limited possibilities to predict when the player decides to make a turn. Figure 3 shows the result this type of prediction will produce when the player makes a movement that cannot be predicted. The correction suddenly moves player's displayed position from $(1, 2)$ to $(2, 3)$ omitting the rotation – a "jump" that cannot be explained with valid movement trajectories and annoys the players.

2. RELATED WORK

As we have seen, latency in online games causes different game clients to disagree on the ordering of events. If we introduce latency hiding techniques, we might also see disagreements on the events themselves. [1] shows that players are in fact aware of these inconsistencies and find those very annoying.

Bouillot and Gressier-Soudan [3] categorize existing approaches to consistency in distributed interactive multimedia applications. For live synchronized media such as games they list two main directions. Both start with the observation that there is a "correct" ordering of events, e.g., the one of the lag free view (other definitions are possible). It is possible to determine this ordering either with synchronized clocks or by introducing logical clocks [4].

The problem is that the clients on the internet are not omniscient so that client A will not know about an event that happened on client B at least until this event had a chance to propagate itself through the network. This means that if the same ordering of events is to be enforced on all clients (pessimistic approach), the clients will for example need to delay processing of local events until all possibly preceding events from other clients have been received ("local lag" [5]). Especially, round-based games like Civilization solve consistency problems successfully with this approach. Here, the game is designed in such a way that for each round, messages from all players are collected before their effects are displayed and the next round starts. In other online games like MiMaze [6] a tolerable amount of local lag can only solve some of the inconsistencies while a significant number of packets will still arrive too late to be processed in order.

Introduction of local lag also means that the player player has to wait before he can see his move executed. While this ensures a globally consistent game view, the price is definitely too high – this delay might make playing the game a torture [7]. Most of the time, this delay is also not even necessary, something optimistic approaches like Timewarp [5] take advantage of. Here, local events are allowed to be processed immediately. However, if a remote event is received that brings up an inconsistency in the event chain, the effects of previously processed events are restored and the events are reapplied in a different order.

A popular variation is the server-centric approach that allows to maintain an absolutely consistent global view at a very low cost. It does not even require synchronized clocks on the clients, the "real" ordering of events is defined as the order in which events arrive at the server. This way the server's view is consistent by definition, and the only problem is to deliver this view to the clients.

On the other hand, there are also approaches that ignore consistency entirely. In P2P scenarios the clients typically process the events in the order they receive them and only correct something when they receive more current data. The hope is that the differences in the local views of the clients will be of minor importance for the game and will not prevent the players from enjoying it. The popularity of games that use this type of approach shows that players can in fact tolerate a reasonable level of inconsistency [8].

3. USER-CENTERED CONSISTENCY REQUIREMENTS

In this paper, we want to introduce an approach that is different from all mentioned above. Instead of focusing on a consistent global state, we want to keep each local state consistent in itself without necessarily requiring it to be equal or even similar to other local views. In our opinion, the most important goal is to make the game appear consistent to its consumer (the player). A perfect game is one where the player cannot tell the difference between a local game (without latency) and an online game.

Below, we define a set of requirements that have to be fulfilled for each local view to reach this goal. All these requirements are defined in terms of user perception and often cannot be directly translated into an implementation, they can be considered however when choosing between different implementation approaches.

3.1 Physical consistency

> *Physical consistency*: Any virtual world event visible to a player has to be compliant with the world's physical model – within user's perception limits.

This is a very basic requirement, meant to ensure that the physical model of the virtual world applies universally to all contained objects regardless of whether those are local or remote. Typically, this means that no object should exceed maximal allowed values for velocity and acceleration. An example that violates this principle is given in Figure 3b) where player A seemingly jumps from position $(1, 2)$ to $(2, 3)$ while rotating at the same time – something that is "impossible" according to game rules. This kind of unexplainable behavior obviously irritates the user and should be avoided whenever possible.

This requirement also rules out situations like Figure 2b), i.e., a player who is displayed as dead should not be able to shoot. "Resurrection" of players should be disallowed as well, e.g., if a shot seemingly resulted in a hit, but the answer from the server told the client that it was in fact a miss, the situation should be resolved in some other way.

Note that the requirement of physical consistency still leaves enough space for correcting actions, for example the physical model can be violated for objects outside user's perception range. One has to be careful however since the user might know about the position of objects he cannot see directly, e.g., something that just moved out of his visual range temporarily. Correcting actions that are visible to the user can also violate the physical model if necessary – as long as those violations are not recognizable. For example, the object can have a velocity slightly above allowed maximum to allow the correction to finish faster.

3.2 State consistency

> *State consistency*: Any change in game state that is important to more than one player has to be signaled to either every player concerned with reasonably low difference in arrival times or to none of them.

Even though the local views are allowed to diverge significantly there is a certain set of parameters that have to be identical for all of them. One typical example is the game score – if it were different on different clients for a longer period of time the players would find out about this inconsistency through game chat or other communication methods. In some games, the score will also influence player's strategy causing behaviour that other players would consider inconsistent.

We call such parameters "important" for lack of a more objective description. Unfortunately it probably is not too easy to identify the parameters for which inconsistencies between local views cannot be tolerated, and it is also highly dependent on the game and the way it is played. In some games, picking up a power-up would be "important" since this power-up becomes unavailable for everybody else. In other games, it is acceptable to let another player pick up the same power-up again if the change did not propagate to him yet – and then it is not too important to keep the list of available power-ups perfectly in sync.

Identifying the set of "important" changes is the most difficult part of maintaining state consistency. Once the necessary changes have been identified, distributing them should be relatively easy – events of this type typically occur relatively infrequently and have rather weak requirements on

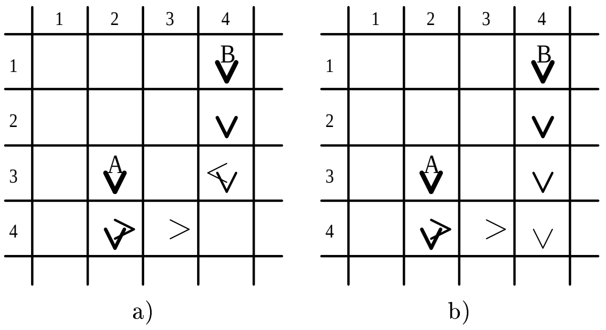

Figure 4: Reacting to an ambush: a) correct reaction, b) unreasonable reaction

latency. Usually, even several seconds delivery time are acceptable.

3.2.1 *World state consistency*

Changes to the virtual world are mostly "important", at least if they affect the decisions players make in the game. For example, if somebody blows up a wall and gives everybody a new shortcut or even access to a new area everybody concerned should know about it as soon as possible. Otherwise, some players might loose a chance others can take simply because of different delivery latency for some important information – this makes the game unfair.

3.2.2 *Player state consistency*

Other than world changes the changes to the players themselves (exact position, damage factors, shape changes, etc.) are usually not subject to the state consistency requirement, i.e. their impact on other players is limited. Typically, this changes only when two or more players are engaged in a direct combat. Here damage caused by hits has a huge impact on strategy in the battle. The most important event is the death of a player of course, it tells the attackers that they can concentrate their efforts on other targets.

3.3 Reaction consistency

> *Reaction consistency*: A player's actions should not appear unreasonable in other players' views unless those actions are also unreasonable in the player's own view.

Competing against somebody who makes stupid mistakes is no fun, and any good game engine should do its best to prevent turning reasonable decisions into those mistakes simply due to technical limitations, above all latency. If player A tries to ambush player B, B should turn around to avoid being killed and maybe attack A himself (Figure 4a). But, if B cannot see the ambush because he receives A's actions with 2 seconds delay his reaction might be the one in Figure 4b), giving A an unjustifiable easy win.

Usually, problems like this occur when the position (or orientation or velocity) of remote players is displayed incorrectly, making the view's owner misjudge the situation. State consistency violations can have the same effect. The system's job in this case is to make sure that differences between local views do not affect a player's judgement in any way that is relevant – at least in most cases.

4. CONCLUSION

As we have seen, most current approaches to ensure consistency in games focus on the global state mostly ignoring the local views. Our suggested approach is the opposite – try to make sure that the local views are consistent since those are what the players see. As long as the players cannot see any inconsistencies it is not even necessary to keep any sort of global state. We formulate three user-centered consistency requirements meant to ensure that the user perceives a game over internet in the same way as an offline game, i.e., all latency-induced artifacts are hidden. Even though these consistency requirements usually cannot be perfectly fulfilled for internet, there is a number of techniques that can be used to achieve satisfactory results. Unfortunately we could not discuss these techniques here due to space limitations.

Of course this paper is only a first step. The consistency requirements need to be formalized further and ideally even become measurable. We have to test them against more game types and probably add some new criteria. Finally more/better ways to meet these requirements are necessary, ideally with measurements on their effectiveness. In particular we will continue our work on improving prediction and convergence algorithms, as well as evaluate and compare their performance.

5. REFERENCES

[1] Manuel Oliveira and Tristan Henderson. What online gamers really think of the internet? In *Proceedings of NetGames '03*, pages 185–193, Redwood City, California, 2003. ACM Press.

[2] Mark Claypool. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks*, 49(1):52–70, September 2005.

[3] Nicolas Bouillot and Eric Gressier-Soudan. Consistency models for distributed interactive multimedia applications. *SIGOPS Oper. Syst. Rev.*, 38(4):20–32, 2004.

[4] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[5] Martin Mauve et al. Local-lag and timewarp: Providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, 6(1):47–57, February 2004.

[6] L. Gautier and C. Diot. Design and evaluation of mimaze, a multi-player game on the internet. In *Proceedings of ICMCS '98*, page 233, Washington, DC, USA, 1998. IEEE Computer Society.

[7] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of NOSSDAV '02*, pages 23–29, Miami, Florida, USA, May 2002. ACM Press.

[8] Tristan Henderson. Latency and user behaviour on a multiplayer game server. In *Proceedings of NGC '01*, pages 1–13, London, UK, 2001. Springer-Verlag.