GLS: Simulator for Online Multi-Player 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

One of the most difficult tasks when creating an online multi-player game is to provide the players with a consistent view of the virtual world despite the network delays. Most current games use prediction algorithms to achieve this, however measuring the effect of different approaches is difficult. To solve this problem we introduce a simulator called GLS that gives us a fully controlled environment and allows large-scale experiments to evaluate different aspects of the algorithms.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia information systems—*Artificial, augmented, and virtual realities*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Benchmarking, Ergonomics, Evaluation/methodology*

General Terms

Measurement, Experimentation, Human Factors

Keywords

games, simulation, latency, GLS

1. INTRODUCTION

Multi-player online games become increasingly popular nowadays. It is simply more challenging to compete against human players than it is to compete against the artificial intelligence the game developers build into their games. One major problem developers of online games have to face is the network latency that makes fast user interactions difficult. Especially first-person shooting games have high demands on accuracy when displaying avatars of remote players. Shooting games require fast reactions from the players, and the players require accurate positions of their opponents without noticeable delay from the game in return.

Usually an approach is used that has been first standardized in Distributed Interactive Simulation (DIS, [2]) standard in 1993. A technique called dead reckoning is used to display realistic movement of remote objects and limit the number of necessary position updates at the same time – the current position is predicted using last received position update simply by assuming that the velocity and acceleration have not changed since. The formula for the position at which DIS will show a remote object is well-known (last term is optional):

$$p = p_0 + v_0\Delta t + \frac{1}{2}a_0\Delta t^2$$

Here p_0, v_0 and a_0 are the position, velocity and acceleration of the remote object as received with the position update. Δt is the time that passed since the position update was sent (difference between current time and the timestamp of the position update). In a similar way the orientation of the remote object can be calculated, with the same assumption that it did not change its angular velocity/acceleration (or at least not too much).

Surprisingly, today the approach proposed by DIS in mid-90s is still used in most multi-player online games with only minor variations. Only a few other prediction algorithms have been proposed, typically with a very limited area of application (e.g. PHBDR [3]). One of the reasons are probably the difficulties of evaluating prediction algorithms properly and measuring the advantages of one prediction algorithm over the other. Most papers are simply guessing what effect something will have on the user, large scale experiments require many players so that those are expensive and rare ([4] are some examples).

2. GAME LATENCY SIMULATOR

Here we introduce our Game Latency Simulator (GLS, [1]) that emulates a network game with an arbitrary number of clients. Using this simulator for evaluation of prediction algorithms gives us several advantages:

- Simulations can be run much faster than at real-time speed
- Using a simplified world model where actions don't have unexpected side effects
- Algorithms can be swapped easily, even from the command line
- Can be run in batch mode with all important parameters specified on command line or in configuration file
- Network is simulated as well so that its characteristics can be adjusted easily
- Avoiding the clock synchronization over network problem, simultaneous events are defined trivially

MM'06, October 23–27, 2006, Santa Barbara, California, USA.
ACM 1-59593-447-2/06/0010.

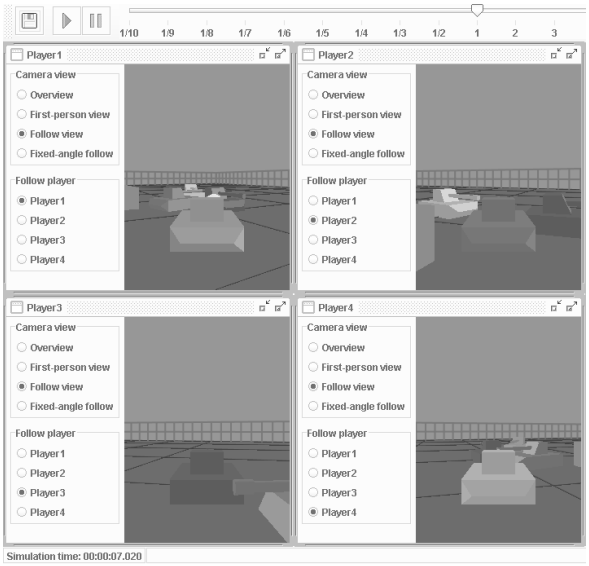

Figure 1: Simulator's graphical user interface

We chose to simulate the game BZFlag in GLS due to its simplicity. It is a first-person shooting game with a very simple goal – drive a tank and shoot as many other tanks as possible. This made it easy to create good computer players and produce results that are relevant. While our results still need to be confirmed with human players, we are confident that we will not see any qualitative difference there.

2.1 Execution modes

Every player has his own view of what is happening in the game based on the position updates sent through the (simulated) network. The simulator's GUI (Figure 1) can show all these views at the same time. It is useful when we need to understand the numbers that we get from our experiments. The main execution mode for the simulator is batch mode however where we can speed up the processing at 1000 times the real-time speed while still processing every frame for every simulated client. The configuration file defines all the necessary parameters, we can also override parameters from the command line, for example the game field size, maximum velocity, number of obstacles and the simulation duration:

```
java batch.Main --field=Field(40,40)
                --maxLinearVelocity=1.5
                --obstacles=3
                --event1=ShutdownEvent(180000)
```

2.2 Architecture

Internally GLS consists of a number of players sharing the same simulation object (Figure 2). The simulation contains a game field, an event manager and a network. The game field usually is a rectangular field with a number of rectangular obstacles. The event manager stores a queue of events with the corresponding timestamps at which these events should be triggered. This is also the component implementing the main simulation loop which only ends when the event queue is empty (as processing some of the events

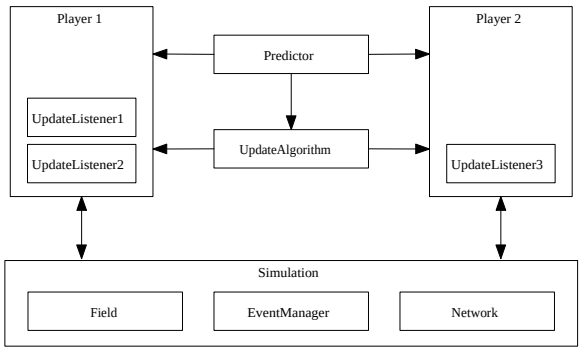

Figure 2: Simulator architecture

always schedules a new event this should only happen after a ShutdownEvent object has been processed). Finally the network is responsible for delivering position updates to other players which is done by scheduling a NetworkEvent in the EventManager for the timestamp at which the position update should be delivered.

The players also hold references to some helper objects. Those are usually the same for all players, a particular player can be made to use a different helper object however. A predictor creates a guess for a remote object's current position based on the previously received position updates, this is the position a player will use for his decision making. For own position updates an update algorithm helper must be consulted to decide whether a position update should be sent. A DIS conform update algorithm will use a predictor to create a prediction based on already sent position updates. A new position update will only be sent if the prediction deviates too much from the actual position. Our default implementation can also take orientation into consideration (pre-reckoning [5]).

The players support an arbitrary number of UpdateListener objects that will be called whenever the state of some object changes in the player's view. This interface is implemented by the GUI to update its display as well as by all evaluation modules which are basically aggregators for this data.

3. REFERENCES

[1] GLS – Game Latency Simulator. `http://www.ifi.uio.no/forskning/grupper/nd/projects/2004/mismoss/gls.html`

[2] IEEE Standard for Distributed Interactive Simulation – Application Protocols. *IEEE Std 1278.1-1995*

[3] Sandeep K. Singhal, David R. Cheriton. Using a Position History-Based Protocol for Distributed Object Visualization. *Technical Report STAN-CS-TR-94-1505*, Department of Computer Science, Stanford University. February 1994.

[4] Tristan Henderson. Latency and User Behaviour on a Multiplayer Game Server. *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, London, UK. 2001.

[5] Thomas P. Duncan, Denis Gračanin. Pre-Reckoning Algorithm for Distributed Virtual Environments. *Proceedings of the 35th conference on Winter simulation*, New Orleans. December 07-10, 2003.