MiMaze, a 3D Multi-Player Game on the Internet

Emmanuel Lety, Laurent Gautier and Christophe Diot
INRIA Sophia Antipolis. [elety|lgautier|cdiot]@sophia.inria.fr. www.inria.fr/rodeo/MiMaze/

Abstract. This paper describes the design, implementation, and evaluation of MiMaze, a distributed 3D multiplayer game on the Internet. The major contribution of this work is to provide the first experimental results for this new type of application. An original fully distributed communication architecture has been designed for MiMaze. It uses multicast communication system based on RTP/UDP/IP, and a distributed synchronization mechanisms to guarantee the consistency of the game, regardless of heterogeneous network delay. This paper provides evaluation results on the Mbone. The experimental evaluation shows that the major problem will not be with real-time interaction, but with scalability.

1. Introduction

This article describes the design and the evaluation of a simple shared virtual world over the Internet. MiMaze is a distributed (i.e. serverless) game that uses an unreliable communication system (MiMaze transmission control is based on RTP [7] over UDP/IP multicast [11]). Multiplayer games are representative of the new generation of Distributed Interactive Applications (DIAs), that also includes Distributed Interactive Simulation (DIS) applications, shared virtual worlds, Air Traffic Control (ATC), cooperative tools, and home interactive applications[3][4]. These applications are expected to represent a major share of the Internet traffic in the next 5 years.

We designed a game with simple rules. However, 3D graphics, sounds, and video give MiMaze a credible complexity in term of rendering and data transmission. Because the architecture is distributed (i.e. each participant computes its own view of the game), we had to design a synchronization mechanism to cope with different transmission delays among the participants. This synchronization mechanism (called bucket synchronization) is analyzed later in this document. It is the minimum functionality required to play a distributed application on the Internet. We also use a dead-reckoning mechanism to recover lost or late information.

For this work, we have chosen to follow some of the DIS [1][2] rules. In particular, we require that any action issued by any participant must be displayed to all participants before 150ms.

This article is structured as follows. Section 2 describes MiMaze. We start with a description of the game and of its functional architecture. Then we describe the VRML graphics and display. We conclude Section 2 with a description of the communication infrastructure. In Section 3, we present and analyze performance measurements. MiMaze resource requirement is analyzed, including CPU load and network bandwidth (realized on the Mbone). The scalability problem is identified. The performance results give us an opportunity to discuss what enhancements are necessary to increase the scalability of distributed interactive applications.

2. Description of MiMaze

The characteristics of MiMaze are very similar to those of DIS applications [1][2] and to shared virtual worlds [12]. MiMaze is different in that the world is very simple (based on simple objects). One reason for choosing a "simple" game is that choosing a more complex game can make it difficult to analyze the game traffic parameters. The virtual worlds characteristics that apply to MiMaze are: real-time interaction between players, large number of participants, 3D graphics with video and audio streams, and high level of dynamicity in group structure and topology.

2.1 MiMaze design characteristics

MiMaze is inspired from iMaze [6]. iMaze is a 2D "Pacman" game with a server-based architecture. Avatars move around in a maze where they try to kill each other.

In MiMaze, players have a 3D representation of their view of the game. The MiMaze architecture is totally distributed, i.e. the state of the game is computed by each participant (there is no server to compute the game state).

In addition to the distributed architecture of MiMaze, we extended the game with MPEG-1 video and audio. With such an extension, MiMaze can be considered to be representative (from a data complexity point of view) of a distributed interactive virtual environment.

The 3D display is realized with the Virtual Reality Modeling Language (VRML[13]; see Figure 1 left). MiMaze runs with the SGI CosmoPlayer 2.x plug-in for Netscape Communicator 4.0x [14]. We use the External Authoring Interface (EAI) of the VRML browser to instantiate, move, and remove dynamically the avatars in the VRML scene. All these operations are done from a Java applet embedded in the same HTML page.

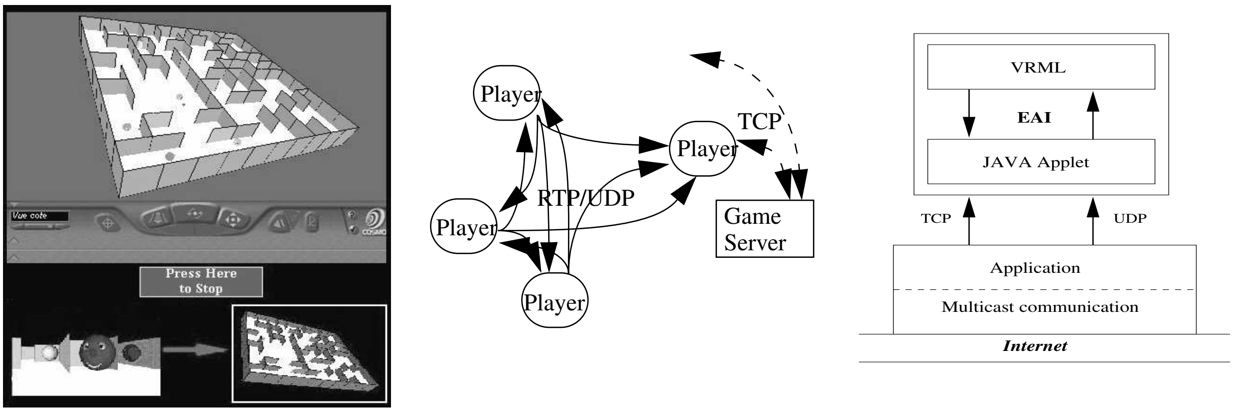


Figure 1 : MiMaze screen shot (left) and system architecture (middle) and display architecture

MiMaze [9] is to our knowledge the only game with a fully distributed architecture using multipoint communication support (a server is only used when a new entity joins a session).

Multicast-based distributed architectures have a number of advantages for interactive multi-participant games, including robustness (e.g., the failure of one entity has no effect on the others) and scalability (e.g., a distributed architecture more easily allows for natural partitioning of game computation as the number of players increases). A discussion of the advantages and disadvantages of a distributed architecture are beyond the scope of this paper and are covered in detail in [9]. We simply note that a distributed game has no server that computes a unique global state (Figure 1 middle). Instead, each entity computes its own local view of the global state of the game using information received from other entities. This locally-computed view is displayed to the local participant. The key challenge then is to provide as timely and as consistent a display as possible. The consistency requirement implies that even though each participant computes its own local view of the games state, a distributed synchronization technique is needed to ensure that the participants compute game states that are as similar as possible at a given time; such a technique is discussed in Section 2.3. Entities must also recover from lost or overly delayed information.

The MiMaze architecture favors real-time interaction at the cost of game consistency. In other words, ***MiMaze communication architecture prefers to display slightly inaccurate avatar positions rather than delayed exact positions***.

2.2 VRML display

We have chosen VRML as a file format for describing the MiMaze virtual world to provide interoperability with other virtual worlds (included with future releases of this game). The user chooses between a set of different external views of the 3D-maze and an internal view where the viewpoint is through the eyes of the user's avatar (Figure 1 left). MPEG1 scenes are mapped on walls of the maze, and participants need only to click on the wall to start the associated movie. MPEG1 scenes are currently played locally (they are not carried on the network).

MiMaze display is made of two major modules: a Java applet, and the VRML browser that provide user navigation interface (Figure 1 right). The Java applet uses the EAI (External Authoring Interface) to interface the game application (including the communication infrastructure) to the VRML browser.

2.2.1 The display interface

MiMaze game and communication system are written in C++. An interface has been designed to send two kinds of informations to the display Java applet (Figure 1 right): The participants "join and leave" information is sent with TCP (i.e. reliable), and all avatar positions and orientations are sent with UDP (i.e. unreliable). Note that the Java applet also manages the video and audio streams.

2.2.2 The Java applet

We have developed a multi-threaded Java applet to (Figure 1 right) (*i*) receive avatar positions and orientations from the UDP socket, (*ii*) receive "join and leave" information from the TCP socket, (*iii*) select the navigation type (the user is outside the maze, or it is in its own avatar), (*iv*) provide access to the VRML nodes through the EAI, and to (*v*) modify the 3D scene. We have also designed a specific scheduling algorithm to maximize the real-time quality of the application.

In MiMaze, the VRML browser is the rendering engine of the application. The frame rate is therefore dependent on the browser and on the 3D graphic card. Using the EAI allows the Java applet and the VRML browser to interact asynchronously. Moreover, the use of two sockets between the MiMaze application and the applet provides optimal application-VRML scene interaction: each time the VRML scene is updated, the most recent positions are used, which maintains the real-time quality of the game.

2.2.3 Event propagation

We decided not to receive any events from the VRML browser (keyboard events are managed at the application level). With this approach, the applet sends events to VRML nodes, but does not have to wait for any information coming from the VRML browser. Moreover, each time the player shoots, the bullet motion must be synchronized with the detonation sound. This synchronization is achieved using a specific script node in the VRML scene. When the player shoots, the Java applet sends a boolean event to this node. In VRML, each time an event is generated, a field value and a time-stamp are delivered to the recipient VRML node. As soon as the script node receives this event, it "routes" its timestamp to the startTime field of the associated AudioClip Node.

2.3 The communication architecture

The two network parameters that influence the game consistency are the packet-loss rate and the variable delays. There are consequently two elements in the MiMaze communication architecture. Distributed synchronization compensates for delivery delay and dead reckoning compensates for missing or late packets. Both mechanisms are described with more details in [9]. To help understanding MiMaze communication infrastructure, it must be explained that each participant sends periodically its full description (made of its position, direction, speed, and angular speed) in a packet called ADU (for Application Data Unit). This description is used to compute the game global state.

2.3.1 Distributed synchronization

Since the network delays are different for any pair of participants on the Internet, synchronization must be introduced to allow ADUs issued at the "same time" to be processed "together" by any participant. Another aspect of synchronization is that all participants should display the same game state at the same time.

In MiMaze, time is divided into fixed length sampling periods and a bucket is associated with each sampling period. All ADUs received by a player that were issued by senders during a given sampling period are stored by the receiver in the bucket corresponding to that interval. When a participant has to deliver an updated global state, it computes all the ADUs available in the "current" bucket.

The synchronization delay is computed on ADU reception to determine in which bucket the ADU content should be stored. The MiMaze synchronization delay is equal to 150ms minus the network delay. In other words, an ADU issued at absolute time t will be rendered at the receiver (if at all) in the interval containing absolute time t + 150ms. If the network delay is more than 150ms the ADU is stored in a past bucket. The bucket mechanism is very close to a playout buffer mechanism [8] used to reduce network jitter effects in packet audio information. The bucket frequency defines the rate at which a new game state is computed (and should be displayed - see Section 3.1). Since human vision perceives continuous motion at a frequency of 25 images per second, we have chosen to compute 25 buckets per second. The bucket frequency is a receiver parameter that should not be influenced by network parameters. The only reason to reduce this frequency would be excessive CPU load.

The bucket synchronization mechanism uses a global clock mechanism to evaluate the delay between participating entities. The clock has to be accurate (a 10ms precision is required [12]) and continuous. In our implementation, we use NTP [5] to synchronize clocks.

2.3.2 Dead Reckoning (DR)

To deliver a complete view of the game, the bucket algorithm requires to have one ADU per participant available in each bucket. However an ADU can be missing for different reasons: it can have been lost by the network; or it can be late. DR is used to replace missing ADUs.

For each missing ADU, the state-computation algorithm goes back to the previous buckets, looking for the most recent ADU received for the missing avatar. Once found, this ADU is dead-reckoned to evaluate the position where the avatar should "most probably" be at the current time. The accuracy of the evaluation depends on the DR algorithm used (there are many possible DR algorithms available), on the "age" of the ADU used, and on game characteristics.

2.4 Related work

Amaze can be considered as MiMaze's ancestor. Amaze was designed by Berglund in 1984 [10] to be played on a LAN, using point-to-point communication. MiMaze and Amaze both have a distributed architecture but manage states differently. Amaze transmits the game state on the network, and maintains replicated copies of the game state.

Distributed games on the Internet are now a real market for private companies. Microsoft, BT, and Intel have their own projects on distributed games, but the approach of these companies has historically been to over-engineer the network in order to maximize the quality of the game; and not to design specific mechanism to increase application robustness on a network.

3. Performance evaluation

In this section, we analyze the scalability of the game, i.e. the impact of the number of participants on the performance of the game. We successively observe the scalability limits from the local resource standpoint (CPU load) and from the network standpoint (network congestion).

3.1 CPU and scalability

In DIAs, the CPU has two main tasks:

- the computation of the global state with data coming from all participants (see the description of the communication architecture). In MiMaze, with no display, more than 30 participants are needed to saturate the CPU[1] (this is mostly due to transmission control mechanisms, and only partially to global state computation).
- the rendering of the global state (including audio, video, and 3D graphics). The rendering module consists in three steps. The applet obtains the game global state description from the application. Then, it uses the EAI to update the 3D scene in the VRML browser. Finally, the VRML browser performs the rendering (this task is asynchronous, and is done partially on the graphic adapter).

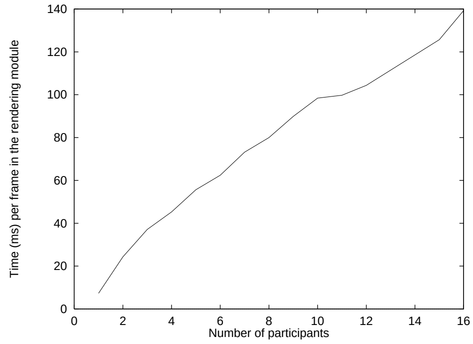
Experiments without display show that the complexity of the global state computation is negligible compared to the rendering. Moreover, game state computation has to be short in order to preserve the real-time properties of the game. Consequently, most of the CPU load comes from rendering. Figure 2 left shows an evaluation of the scalability of the rendering module. We observe that the time needed to display a single global state grows linearly as the number of objects to display increases. Figure 2 right reveals that after a few number of participants, it becomes impossible to display games states continuously to the player. Note that using a high level language as VRML and using the EAI to access the VRML browser contribute to explain MiMaze’s poor performance. Adding more CPU resources would improve game performance but would not resolve the scalability problem.

3.2 Internet experiments

The performance metric we have chosen to express MiMaze’s consistency is the "drift distance". It computes the distance between the real position of an avatar (computed with no network delay) and the position displayed by remote participants. In a perfect situation, this distance would be zero.

1. Performances were evaluated on a PC Pentium Pro 200Mhz, Windows NT with a Diamond Fire Open GL 1000 Pro.

We first measured the scalability of MiMaze locally, using a non-loaded Ethernet. In this situation, network losses should have been insignificant. Figure 3 left shows this is not the case. On Figure 3 left, the mean network delay, the mean loss rate, the mean game consistency and the standard deviation of the game consistency are shown for varying numbers of participants.



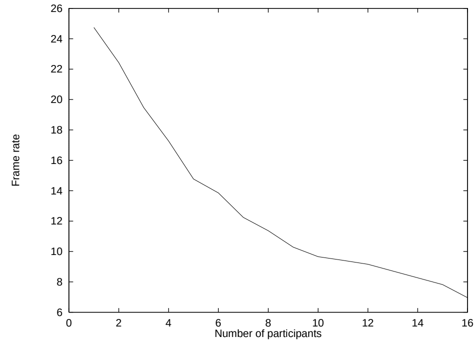
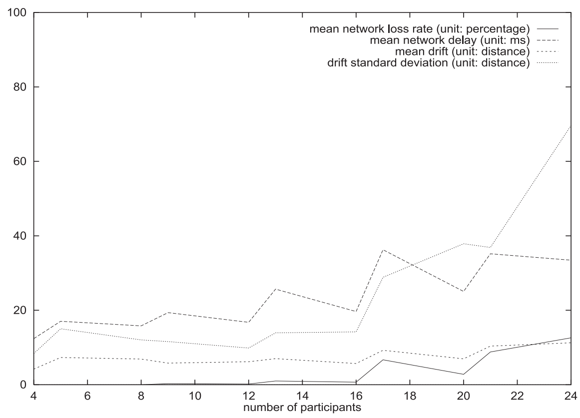


Figure 2 : MiMaze CPU scalability

If the network delay and the mean consistency remain quite stable up to 20 participants, the network loss rate dramatically increases. This behavior is not due to the bandwidth used by MiMaze (approximately 16 kbits per second and per participant), but to the high frequency of short ADUs that are periodically (and synchronously) sent on the Ethernet, increasing the number of collisions on the network. For 20 participants, the network loss rate reaches 15%; and the consequences of these losses on the consistency's standard deviation is dramatic (exponential growth).

Considering these scalability limits on a local-area network, we decided to have more experiments on the Internet (more precisely on the Mbone, which is the multicast backbone overlay of the Internet). Figure 3 gives a snap-shot of what is the behavior of MiMaze with 5 participants on the Mbone.



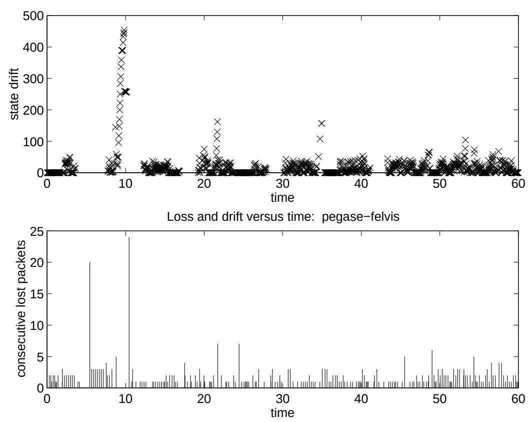


Figure 3 : Scalability on a Local Area Network (left) and consistency observed on the Mbone (right)

Figure 3 right contains two plots. The upper graph shows the drift (or distance) between site A's actual avatar position and a remote site (B's) estimate of that position as a function of time. The lower graph in Figure 3 right plots the number of consecutive packet losses, i.e., consecutive ADUs not received from A by B. The most important observation, in the context of this paper, is that when the loss rate is small, the drift is generally very small. In the best case, there is no drift (i.e., the estimate of a remote avatar's position is exactly correct). This occurs during intervals of time around 2, 23 and 36 seconds. We note that for bursts of 3 to 5 consecutive losses, our DR algorithm succeeds in estimating the remote avatar's position with limited error (specifically, with an error of less than 50 units in 90% of the cases). An avatar being 64 units wide and moving at a speed of 32 units every 40ms, a 50 units drift is not sensitive to the player and does not affect its behavior in a game session.

4. Conclusion

To our knowledge, MiMaze is the first 3D multi-player game to be designed with a distributed architecture, i.e., serverless. MiMaze opens the door to a new generation of interactive applications on the Internet. Another major contribution of this work is to demonstrate the efficiency of distributed architectures based on multicast communication. We have identified scalability to be the major problem in DIAs. Despite the fact that in our experiments, the scalability is limited first by rendering, we believe rendering will not be the key issue in the future:

- The rapid progress in computer horsepower and 3D graphic cards increases everyday the rendering performance.
- The game state computation time must be kept small compared to the display frequency. This is a design principle when it comes to design real-time interactive applications.
- DIAs are many-to-many applications (and not one-to-many applications) with a strong potential to create network congestions. Compared to CPU, network bandwidth is a scarce, shared, and expensive resource.

To overcome the problem of scalability, we have identified two approaches:

- In MiMaze, each player does not need to receive information from all the other players, especially not from those who are far away with no possible interaction. Dividing the maze in many cells and assigning to each cell a multicast group should significantly reduce the amount of useless packets pending in the network. The problem is that the IP multicast model needs to be improved before participant sub-grouping can be deployed on the Internet.
- Object based encoding, and in particular MPEG4 streaming, is also expected to increase scalability. Hierarchical object encoding (such as provided in MPEG) will allow to efficiently transmit data flows while taking into consideration network dynamics. Such encoding techniques should be defined in collaboration between the network research community and the multimedia research community.

It should be noticed that the first technique both increases network and CPU scalability by reducing the number of objects to be processed and displayed to the minimum required from an application standpoint. The second approach will minimize the amount of data transmitted on the network, with very limited influence on the CPU scalability.

5. References

[1] IEEE Standard for Distributed Interactive Simulation -- Application Protocols. IEEE Std 1278. 1995.

[2] IEEE Standard for Distributed Interactive Simulation -- Communication Services and Profiles. IEEE Std 1278.2-1995.

[3] S. Seidensticker, W. Garth Smith, and M. Myjak. "Scenarios and Appropriate Protocols for Distributed Interactive Simulation". Working Internet Draft <draft-ietf-lsma-scenarios-01.txt>. March 1997.

[4] J. M. Pullen, M. Myjak, and C. Bouwens. "Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment". Working Internet Draft <draft-ietf-lsma-limitations-01.txt>, March 1997.

[5] D. L. Mills. "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC-1305, March 1992.

[6] J. Czeranski and H-U. Kiel. "Softwarepraktikum Netzwerkprogrammierung unter Unix am Beispiel des Spiels". 1993/94, http://www.tu-clausthal.de/student/iMaze/.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications". RFC-1889, January 1996.

[8] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks". Proceedings of Infocom '94, Toronto, Canada, pp. 680-688, April 1994.

[9] L. Gautier and C. Diot. "Design and evaluation of MiMaze, a Multiplayer Game on the Internet". IEEE Multimedia System Conference. Austin. June 28 - July 1, 1998.

[10] E. Berglund and D. R. Cheriton. "Amaze: a multiplayer computer game". IEEE Software. 2(3):30-39, May 1985.

[11] S. Deering. "Host Extensions for IP Multicasting". RFC 1112. 17. August 1989.

[12] R. C. Waters. "Time synchronization in Spline". MERL report TR96-09. April 1996.

[13] ISO/IEC 14772-1:1997. The Virtual Reality Modeling Language Specification "VRML97". http://www.vrml.org/Specifications/VRML97.

[14] Cosmo Player 2.1 VRML Browser. http:// cosmosoftware.com.