Universidad de las Ciencias Informáticas

Entornos Virtuales

Facultad 5

Catalogación de técnicas de reducción de los efectos negativos de la latencia en simulaciones y juegos virtuales.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Ernesto Caram González

Tutor: Ing. Frank Puig Placeres

Julio 2008

“En la ciencia todo el crédito va al hombre que convence al mundo de una idea no al que la concibió primero.”

William Osler.

Declaración de Autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2008.

Ernesto Caram González **Frank Puig Placeres**

_____ _____

Firma del Autor Firma del Tutor

Datos de Contacto:

Ing. Frank Puig Placeres

Graduado en Ingeniería en Ciencias Informáticas, 1 año de graduado. Profesor en la Facultad 5 de la Universidad de las Ciencias Informáticas. 10 años de experiencia. Coautor de los libros SHADERX5, Game Programming Gems 5, Game Programming Gems 6 y AI Wisdoms 4.

E-mail: fpuig@uci.cu.

Agradecimientos:

El autor es deudor de muchas personas que han contribuido a realizar este trabajo mejor. En primer lugar destacar la preocupación del tutor que siempre estuvo al tanto del proceso de realización y desde el primer momento me brindo su apoyo y conocimiento acerca del tema. A mi colega Darwin, sin su ayuda desinteresada todo hubiera sido más difícil.

De manera particular quisiera agradecerles a mis padres, mi madre que brindo su hombro y su apoyo. Mi padre que no estuvo presente de cuerpo, pero su ejemplo es imperecedero. A mi tío su fe en mi y sus ánimos para seguir adelante, apoyándome en todo lo que fuera necesario. Si hoy estoy aquí es en parte gracias a ellos.

Dedicatoria:

Agradecer a todas las personas que brindan desinteresadamente su conocimiento y contribuyen al mejoramiento humano.

Resumen:

Mitigar los inconvenientes que causan los problemas inherentes a las redes de computadoras ha sido una tarea ardua que muchos han enfrentado. Variadas soluciones, tantas como inconvenientes. En muchos casos una misma técnica es necesario adaptarla para situaciones específicas.

De interés particular resultan las técnicas de predicción del lado del cliente, con el Dead Reckoning como exponente fundamental y las técnicas de optimización del la cantidad de información que se envía para cada actualización ejecutada por el servidor (partición del espacio, relevancia, priorización, compresión delta,..).

El objetivo del presente trabajo es desarrollar un catálogo con las descripciones y en algunos casos ejemplos de código de las diferentes técnicas predictivas y de optimización. Se propone como un material de consulta para la mejor comprensión y futura implementación de los mismos. Podría constituir un material de apoyo para la implementación de los módulos de red para futuros juegos y simulaciones.

Para satisfacer los objetivos propuestos se realizó el estudio de los temas: limitaciones de las redes, topologías de red básicas más empleadas, priorización, extrapolación, interpolación y algunos juegos que han implementado estas técnicas, así como otros asuntos de relacionados.

Palabras claves:

- Técnicas de predicción.
- Dead Reckoning.
- Optimización del envío de la información.

Summary:

To mitigate the inconveniences that cause the inherent problems to the nets of computers has been an arduous task that many have faced. Varied solutions, so many as inconveniences, in many cases oneself technique is necessary to adapt it for specific situations.

Of particular interest they are the prediction techniques on the side of the client, with the Dead Reckoning like fundamental exponent and the techniques of optimization of the quantity of information that it is sent for each upgrade executed by the server (partition of the space, relevance, priorization, delta compression...).

The objective of the present work is to develop a catalogue with the descriptions and in some cases, examples of code of the different predictive techniques and of optimization. Is intends as a consultation material for the best understanding and future implementation of the same ones. It could constitute a support material for the implementation of the net modules for future games and simulations.

To satisfy the proposed objectives was carried out the study of the topics: limitations of the nets, basic most used net topologies, priorization, extrapolation, interpolation and some games that have implemented in some way these techniques, as well as other matters of interest.

Key Words:

- Predictive techniques.

- Dead Reckoning.

- Optimization of the quantity of information.

Tabla de Contenidos:

Introducción

Los juegos de video han sido siempre un tema controvertido, sin embargo es innegable su impacto más que significativo en el comercio mundial de software, además de su uso cada vez más extensivo con fines educativos y de creación de habilidades.

Uno de los principales problemas al implantar aplicaciones de virtuales multiusuario (como los juegos de video en línea, con servidores dedicados (Dedicated Server Hosting)) , es el tiempo que se emplea en la generación y actualización de imágenes del mundo virtual en cada una de las computadoras de los usuarios participantes, para proporcionarles la ilusión de que todos están viendo los mismos objetos y que están interactuando los unos con los otros dentro de un único y uniforme espacio – tiempo . Por lo general, este tiempo es demasiado en este tipo de aplicaciones, debido principalmente a la cantidad de información que debe viajar por la red y todo lo que ello implica (latencias altas, congestión de red, disminución del ancho de banda).

Podemos pensar que este problema se va solucionando con la evolución de las tecnologías y el cada vez más novedoso y eficiente hardware de red, pero esto dista de ser cierto. Los software que definen las aplicaciones virtuales multiusuario también evolucionan en consonancia con la ley de Moore, y por tanto el ancho de banda y demás recursos de red nunca serán lo suficientemente grandes. Las técnicas de predicción y convergencia permiten aligerar este problema.

Se plantea la necesidad de disponer de una forma rápida y efectiva de conocer que técnicas y algoritmos de predicción y convergencia, se deben implementar en los futuros juegos para redes, que se desarrollen en la Facultad 5.

La **situación problémica** expuesta con anterioridad deviene en necesidad de dar respuesta al siguiente **problema científico**:

¿Cómo evitar la pérdida de la continuidad lógica y la consistencia entre los datos que definen la percepción de simulaciones y juegos?

El **objeto de estudio** de la presente investigación son las diferentes técnicas de reducción de impactos en la calidad de la percepción de los jugadores producto de la latencia.

Este catálogo pudiera ser una fuente más concisa y asequible para los proyectos de la facultad que desarrollan juegos para redes, y necesariamente deben implementar dichas técnicas.

Para el logro del objetivo de la investigación y teniendo en elementos anteriores, se plantean las **tareas científicas** siguientes:

- Investigar las arquitecturas de redes con las que se han implementado juegos y simulaciones virtuales.
- Describir y Caracterizar las técnicas más comunes para reducir los impactos de latencia
- Elaborar un demo ejemplificando alguna de las técnicas anteriores.

Objetivo General:

Presentar un catálogo con las diferentes técnicas de reducción de impactos en la calidad de la percepción de los jugadores producto de la latencia.

Objetivos Específicos:

- Realizar el catálogo de técnicas en las que se describan las ventajas y desventajas de cada una, así como las situaciones donde debe aplicarse.
- Crear una aplicación que sirva como ambiente controlado para realizar pruebas a las diferentes técnicas de optimización, presentándose finalmente 2 de estas técnicas

Tareas de la investigación:

- Investigar y caracterizar los diferentes factores que limitan el rendimiento de la red
- Investigar las arquitecturas de redes con las que se han implementado juegos y simulaciones virtuales.
- Describir y Caracterizar las técnicas y algoritmos más comunes para reducir los impactos de latencia.
- Elaborar un demo de algunas de las técnicas anteriores

Capítulo 1: Factores que imponen limitaciones al juego en red.

En mayor o en menor medida, los efectos negativos de la latencia en los juegos en red aparecerán, y su eliminación es imposible, ya que la latencia es simplemente, el tiempo que requieren los paquetes de información para llegar desde una computadora a otra a través de la red. Esta propiedad se ve influenciada mayormente por el recorrido realizado por los paquetes en transición entre computadoras.

La latencia generalmente no es demasiado problemática al jugar juegos de poco movimiento, basados en turnos, como el ajedrez. Por otra parte, los juegos que requieren niveles altos de interacción entre jugadores, conocidos por las siglas en ingles MMORPGS *(massive multiplayer online role-playing games),* MMORTS *(Massively multiplayer online real-time strategy)* y MMOFPS *(Massively multiplayer online first-person shooter)*, pueden ser gravemente corrompidos por las latencias normalmente experimentadas. Esto se vuelve problemático, ya que los jugadores esperan que los tiempos de respuesta y consistencia de los juegos distribuidos se aproximen a los del mismo juego cuando no se juega en red.

1.1- Limitaciones:

Los juegos para redes enfrentan principalmente dos factores limitantes:

- **El ancho de banda.**
- **La latencia.**

Estos recursos se refieren a las cualidades técnicas subyacentes de la red e imponen restricciones físicas, que las aplicaciones no pueden superar y que se deben considerar en su diseño.

El ancho de banda de la red está definido por la cantidad de información que la red puede transportar en un tiempo dado. En contraste, la latencia de la red es el tiempo que requiere un paquete de datos (supóngase uno infinitesimalmente pequeño) para viajar a través de la red de remitente hasta el receptor. En resumen, la latencia representa el retraso acumulado para un determinado enlace de red.

La falta de un ancho de banda adecuado y la latencia que imponen las redes actuales significan restricciones críticas al rendimiento óptimo de la acción en los juegos en línea. El ancho de banda

limita la cantidad de información a manipular y la latencia limita la jugabilidad fluida y consistente, de un juego. Por ejemplo, el ancho de banda limita el número máximo de jugadores que un juego admite (dado que cada jugador adiciona nuevos datos al tráfico de red) y las latencias altas causan un grave retraso entre la acción iniciada y la vista experimentada por el jugador.

Los problemas del ancho de banda a menudo pueden resolverse después de que un juego está escrito (por ejemplo, comprimiendo y priorizando datos, limitando el número de jugadores,...). Sin embargo, los problemas de latencia deben ser manejados desde el diseño de este, preferentemente antes de que el juego sea implementado.

Cuando medimos el ancho de banda y la latencia, lo realizamos a lo largo de un enlace de comunicación a una red. En un juego multijugador, hay típicamente muchos enlaces de red, y cualquier debate acerca del ancho de banda y la latencia merece alguna atención con respecto a la topología de la red.

1.2- Topologías de red:

Existen muchas y variadas topologías de red, a continuación se exponen las más comúnmente empleadas en juegos y simulaciones. Si implementamos una red en forma de anillo, por ejemplo, como se muestra en Figura 1, la latencia desde el jugador 1 hacia el 4 es la suma de todas las latencias a lo largo de todos los enlaces de red entre ellos.

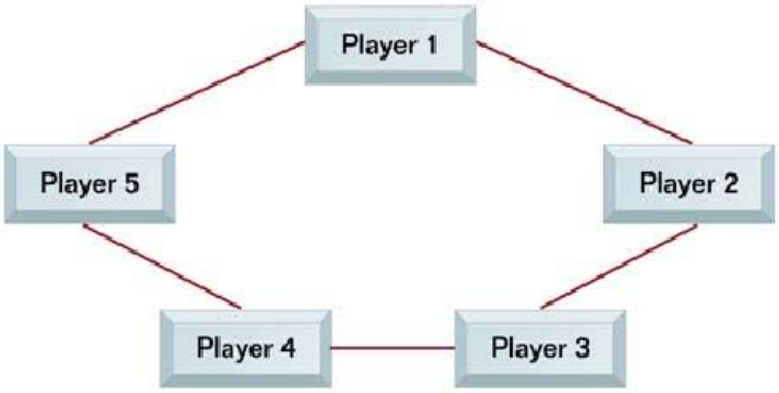


Figura 1: Topología de Anillo.

El ancho de banda es de modo semejante limitado por todos los enlaces entre fuente y destino. Otra posible configuración de la red es la topología todos-a-todos, mostrado en Figura 2.

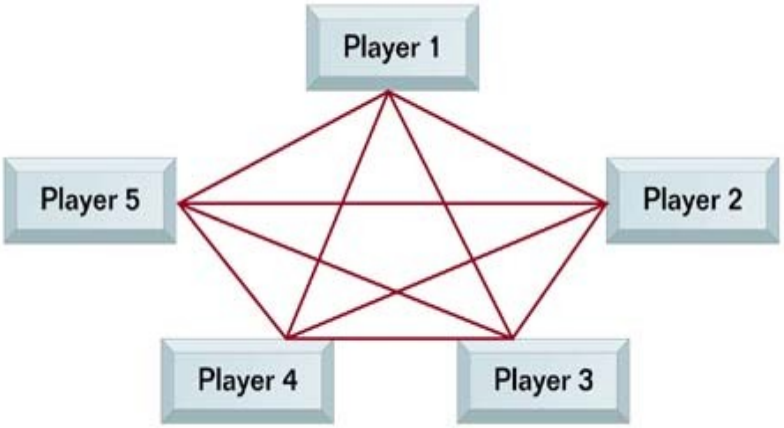


Figura 2: Topología todos a todos.

Esta configuración generalmente logra más bajas latencias entre jugadores, pero a costa de un mayor uso del ancho de banda y una complejidad agregada a las comunicaciones y al control y administración del juego. Generalmente, será mejor en términos de ancho de banda y latencia una configuración en estrella en la cual todos los nodos de juego se conectan a un servidor centralizado, como el mostrado en Figura 3

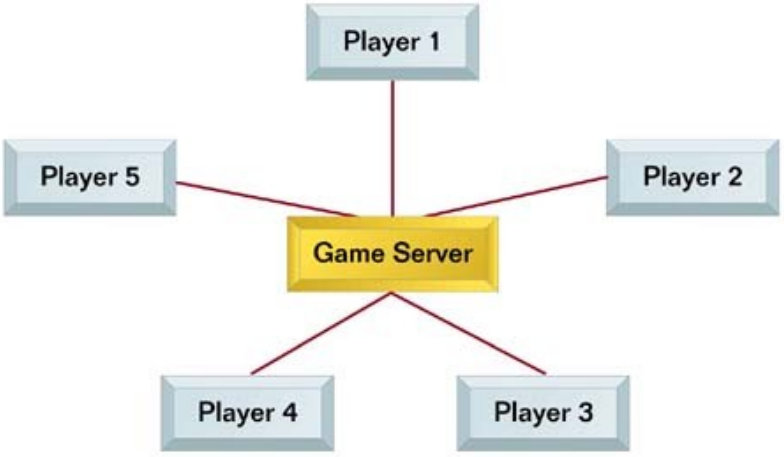


Figura 3. Topología cliente servidor.

En esta configuración, el servidor mantiene una conexión para cada nodo del juego y reenvía los datos enviados por un nodo hacia otro. En una comunicación jugador a jugador, el ancho de banda está limitado por sólo dos conexiones de red y está más probablemente limitado por los módems o puntos de red de los jugadores. Las latencias entre cualquier de los dos jugadores están de modo semejante, dependientes a sólo dos conexiones de red y no son, como en la configuración de anillo, iguales a la suma de las latencias a lo largo de varios enlaces serializados.

Una modificación muy común actualmente a esta topología es la Red de Servidores donde existe más de un servidor interconectado.

Aquí, el gráfico de comunicación puede pensarse de como una red par-a-par de servidores a la que se conecta un conjunto de nodo-servidor. Un nodo se conecta a un servidor local que es conectado a los servidores remotos y, a través de ellos, a los nodos remotos. Podemos extender jerárquicamente la red de servidores para que los restantes servidores hagan de nodos de otro servidor superior jerárquicamente y así supervisar y controlar mejor la estructura. Esta arquitectura reduce los requisitos de rendimiento impuestos a un servidor solitario. En consecuencia, esto proporciona una alta escalabilidad pero aumenta la complejidad al monitorear y controlar el tráfico de red.

En el caso común en el cual todos los jugadores se conectan a Internet, si el servidor puede ser colocado entre (en espacio de latencia) todos los jugadores, la red estrella logra condiciones cercanas a las óptimas, de latencia.

1.3- Ancho de banda:

El ancho de banda se refiere a la capacidad de transmitir una línea de comunicación tal como es por la red. En otras palabras, el ancho de banda es la proporción de la cantidad de datos transmitidos o recibidos por unidad del tiempo. En una red amplia (WAN), el ancho de banda puede extenderse desde los diez Kbps (kilobits por segundo) de módems de marcado hasta 1.5 Mbps de T1 y 44.7 Mbps de T3. En una red de área local (LAN), el ancho de banda es mucho más grande, extendiéndose a partir del 10 Mbps a 10 Gbps. Sin embargo, las LANs tienen un tamaño limitado y soportan un número limitado de usuarios, mientras que los WANs permiten conexiones globales.

Además de cuán a menudo y cuán grandes son los mensajes que se envían, el ancho de banda depende de la cantidad, distribución de usuarios y de la técnica de transmisión.

Asumiendo una configuración de la estrella (que es la topología más comúnmente empleada en los servicios de juego en línea), el ancho de banda de la red está limitado primordialmente por la velocidad del enlace usado por cada jugador y la calidad de esta. Al jugar por la Red utilizando TCP o UDP, se sobrecarga algo más, por el encabezado de cada paquete de red.

Cálculos simples harán ver que el ancho de banda es un recurso escaso. Por supuesto, los datos distribuidos a través de un servidor inteligente (en cuanto al envío de la información) podrían aliviar la carga del ancho de banda específica del juego, filtrando cuáles mensajes son enviados a que nodos. En todo caso, cuántos y cuáles datos son enviados y recibidos debe ser definidos, monitoreados y controlados por el servidor.

1.4- Latencia:

La latencia de una conexión de red indica el tiempo que demora un mensaje en llegar de un nodo a otro en la red. Además, la variación de la latencia en el tiempo (jitter en ingles) es otra característica que afecta a las conexiones de red. La latencia no puede ser eliminada totalmente. Por ejemplo, la propagación a la velocidad de luz de la señal eléctrica se retrasa de 25-30ms al cruzar el Atlántico. Por otra parte, el enrutando, encolando, y manipulación de paquetes aumentan docenas de milisegundos la latencia total, que es en parte debido a los nodos que procesan el tráfico.

Es necesario observar que la latencia y el ancho de banda no están necesariamente relacionados: Podemos tener una red de un gran ancho de banda que tenga una alta latencia y viceversa.

Para los sistemas interactivos a tiempo real como los juegos de la computadora, la regla de oro es que las latencias entre 0.1 y 1.0 s son aceptables. Por ejemplo, para las Simulaciones Interactivas Distribuidas (DIS en ingles) la norma de uso en las simulaciones militares es que la latencia de la red debe ser menor de 100 ms(1). Latencia afecta el rendimiento del usuario: Un control continuo y fluido solo es posible cuando la latencia no excede los 200 ms, a partir de este punto se vuelve cada vez más inoportuna; aunque el umbral depende del tipo de juego. En un juego de estrategia a tiempo real, una latencia más alta (incluso a 500 ms) puede ser aceptable, con tal de que no sea un juego de mucha acción(2). De forma interesante, los experimentos en los ambientes virtuales colaborativos han rendido resultados similares (3).

Hay muchas fuentes diversas de latencia de la red. Las latencias de ida y vuelta pueden variar

desde un mínimo de entre 1 y 10 mseg a 120 segundos o más, dependiendo de una buena cantidad de variables, debido a la naturaleza poco confiable e intrínsecamente heterogénea de la Internet.

Realizar ping es una buena manera para medir la latencia entre dos maquinas, ya que se mide la latencia de ida y vuelta (es decir, bidireccional).

En la mayoría de aplicaciones de computación, la velocidad de la computadora sobrepasa los tiempos de respuesta de humanos, pero vemos que ahora, hablamos de una décima parte de un segundo, lo cual raya con la posibilidad de percepción humana.

Algo a tener en cuenta es el tiempo real gastado en transmitir y recibir un mensaje de cualquier tamaño significativo. Por ejemplo, si medimos la latencia como el tiempo que transcurre entre que se recibe el primer byte de un mensaje y el último byte recibido, de hecho toma algún tiempo diferente de cero entregar el contenido del mensaje, además de que la velocidad de la conexión casi nunca es la máxima soportada por esta.

Otro dato interesante: ¿Por qué frecuentemente se producen en la práctica valores de latencia de casi el doble? La mayor causa de latencias tan desmesuradas son los routers. Sin entrar en demasiado detalle, basta decir que los routers comúnmente pueden causar centenares de mseg de latencia adicional. Los routers acumulan los datos en buffer antes de reenviarlos, agregando una penalización de latencia. Un router frecuentemente descarta paquetes cuándo es sobrecargado, así incurriendo, para TCP, en penalidades de latencia de ***3t+s***, dónde la ***t*** es la latencia de una conexión, donde su principio y fin son dos módems y ***s*** es el retraso por intercambio de mensajes(4). ***3t+s*** fácil y conservadoramente, puede estar en el rango 450msec. Las redes mal configuradas, comúnmente le añaden otro 10% de paquetes descartados.

1.5- Superando las Latencias:

Los juegos de acción rápida, apropiadamente diseñados, pueden ser bastante bien jugados en Internet, incluso para las latencias altas (ej.: 250 mseg). Diseño es la palabra clave. La tolerancia para 250msec de latencia, realmente necesita ser diseñada como parte del modelo de comunicaciones del juego si se quiere ofrecer una buena experiencia de juego.

Una visión acertada y una variedad de técnicas pueden ayudar a compensar y disimular las latencias propias de Internet.

La latencia de la red provoca que los jugadores perciban el escenario imperfectamente sincronizado, pero esto es, hasta cierto punto aceptable, por lo menos para períodos cortos de tiempo. Específicamente, es aceptable para jugadores diferentes tener percepciones diferentes del mundo, e incluso es aceptable no conciliar todas las diferencias. Sin embargo, si es necesario conciliar las diferencias importantes para los jugadores que interactúan, por ejemplo, el proyectil que un jugador dispara a otro, etc.). En este documento se expondrán diferentes técnicas para lograr esto.

1.6- ¿TCP o UDP?

1.6.1- TCP:

TCP probablemente es el protocolo más comúnmente utilizado en aplicaciones generales, pudiendo ser encontrado en muchas aplicaciones, como HTTP, POP, SMTP, etc. TCP es un protocolo que garantiza que el receptor recibirá exactamente lo que el remitente envió, no habrá ningún error, estará en el orden correcto, todo trabajará simplemente bien.

¿Cómo exactamente TCP garantiza esto? Si un paquete se pierde, entonces TCP lo reenvía. Esto significa que todo llegara correctamente, no que llegara rápidamente. Si hay muchas pérdidas de paquetes, el mensaje global podría tardar bastante en llegar. Por eso no es común su utilización en aplicaciones que necesiten tiempo real.

Lo que TCP realmente realiza es disminuir la velocidad de envío en un 50% siempre que existan paquetes perdidos, para aumentarla gradualmente cuando se reciben con éxito. Esto evita la congestión, pero un par de paquetes perdidos realmente pueden reducir dramáticamente la velocidad. Otra casa a notar sobre TCP es que es un protocolo basado en conexión. Un programa abre un puerto que se comunicara con otro puerto abierto por otro programa, probablemente en otra computadora. TCP es evidentemente útil en aplicaciones donde el tiempo de entrega y recepción no son críticos. En fin, para cualquier cosa que sea de vital recepción, pero que no preocupa su velocidad de entrega y recepción.

1.6.2- UDP

UDP es similar a TCP solo en que es protocolo para enviar y recibir paquetes por la red, Existen dos grandes diferencias. Primero, no establece conexiones. Esto significa que un programa puede enviar paquetes a otro, pero ése es el principio y fin de su relación. El receptor podría enviar alguna confirmación al remitente y el remitente podría enviar alguna respuesta, pero nunca hay una conexión sólida como tal. Un nodo o servidor simplemente podría dejar de enviar y/o recibir paquetes.

UDP también es diferente de TCP en que no proporciona garantías de ningún tipo de que el receptor recibirá los paquetes en el orden correcto; no hay ninguna garantía de hecho, de que los paquetes llegarán en lo absoluto. Todos lo que se garantiza es que el contenido de los paquetes, de llegar, llegará exactamente como fueron enviados, sin corrupción de los datos.

UDP es mucho más rápido que TCP, porque no hay ningún encabezado para verificación de errores en los paquetes. Por esta razón, los juegos que requieren transmisión rápida constante de datos utilizan UDP. Cuando una información se envía constantemente, es aceptable que algunos paquetes se extravíen en el camino, ya que nuevos paquetes llegaran a solucionar el embrollo.

Existen varias bibliotecas de Red (como RakNet) que crean su propia capa de fiabilidad encima de UDP, haciéndolo en la práctica y para las necesidades de un juego, tan fiable como TCP, pero con algunos de los beneficios de velocidad del protocolo de UDP. Normalmente los programadores preferían utilizar el ineficiente (en cuanto a velocidad) TCP, cuando existía la necesidad de cierta fiabilidad, ya que implementar una capa de fiabilidad encima del UDP revestía una complejidad agregada. Esto hoy día no es tanto así, ya que bibliotecas, como la anterior mencionada se ocupan de este detalle eficazmente.

UDP presenta diferente variantes que responden a diferentes necesidades:

1.6.2.1- No Fiable:

Los paquetes son enviados por UDP puro. Pueden llegar desordenados, o no llegar .Es mejor para datos que no son importantes, o que se envía muy a menudo, así si algunos paquetes son perdidos, la llegada de nuevos compensaría está perdido.

Ventajas: Estos paquetes no necesitan ser reconocidos por la red, ahorrándose 50bytes al no ser necesario el encabezado de confirmación. El ahorro en ancho de banda podría llegar a ser sustancial.

Desventajas: No hay orden en los paquetes, los paquetes podrían nunca llegar, estos paquetes son los primeros al ser descartados en los enrutadores si son sobrecargados.

1.6.2.2- No Fiable secuenciado:

Es similar al No Fiable puro, pero con la particularidad de que solo los paquetes más nuevos serán aceptados por el receptor. Los antiguos son ignorados.

Ventajas: las mismas que las del No Fiable puro, además de que queda eliminada la preocupación de que los paquetes antiguos trastoquen los datos.

Desventajas: muchos paquetes podrían ser descartados por ser, al fin y al cabo UDP, presentando las mismas desventajas que el No Fiable puro.

Nota: puede ser necesario transmitir en alguna de sus variantes fiables para detectar conexiones pérdidas, de lo contrario puede ser necesario implementar la detección de conexiones perdidas manualmente.

1.6.2.3- Fiable:

La fiabilidad de los paquetes de UDP es monitoreada por una capa de fiabilidad implementada por encima del protocolo, para asegurar que alcanzan su destinatario.

Ventajas: los paquetes llegaran eventualmente, de una forma u de otra.

Desventajas: las retransmisiones y confirmaciones pueden requerir un ancho de banda significativo, aunque normalmente menores que si fuera TCP. Los paquetes pueden ser retrasados si la red está muy ocupada. Los paquetes podrían llegan desordenados.

1.6.2.4- Fiable ordenado:

Los paquetes fiables ordenados son monitoreados por la capa de fiabilidad para asegurar que el arribo de los mismos hacia su destino sea ordenado.

Ventajas: Los paquetes llegaran en el orden en que fueron enviados. Es el más fácil de utilizar, ya que no existen las preocupaciones por evitar los inconvenientes que implican los paquetes desordenados o perdidos.

Desventajas: retransmisiones y confirmaciones aumentan los requerimientos de ancho de banda. Los paquetes podrían retrasarse si la red está sobrecargada. Un paquete tardío podría retrasar a mucho otros que arribaron antes, teniendo como resultado retrasos y latencias. Sin embargo, esta desventaja puede ser mitigada con el uso inteligente de ordenamiento de flujo.

1.6.2.4- Fiable secuenciado:

Los paquetes fiables ordenados son monitoreados por la capa de fiabilidad para asegurar que el arribo de los mismos hacia su destino sea secuenciado.

Ventajas: Se obtiene fiabilidad para los paquetes UDP, el orden del ordenamiento de paquetes, además de no tener que esperar por paquetes viejos retrasados. La cantidad de paquetes que arribaran con este método será mayor que con el No Fiable secuenciado, y serán mejor y eficientemente distribuidos.

Desventajas: Gasto y utilización intensiva del ancho de banda de la red por el uso del encabezado de confirmación de UDP, y de asegurarse de que los datos que han sido descartados sean entregados en la secuencia correcta.

Capitulo 2: Técnicas Propuestas.

2.1- Técnicas de Agrupamiento:

Los juegos virtuales en red presentan una complejidad agregada a la hora de distribuir sus datos. Los desarrolladores deben considerar que información los otros jugadores necesitan, cuando y cómo hacérselas llegar eficiente. La forma más fácil de realizarlo es difundir todo lo que se pueda considerar importante por la red, para mantener a todo el mundo sincronizado. Esta alternativa rápidamente consume el ancho de banda disponible. La otra alternativa seria enviarle a cada jugador solo la información que realmente necesita. Ésta otra alternativa realiza un uso eficiente de los recursos de red, pero decidir a quién se le enviaran los diferentes tipos de información, y a quien no, pueden reducir drásticamente el poder de procesamiento del ordenador.

Existe una técnica, que podría considerarse como un término medio entre ambas opciones anteriormente descritas, denominada *agrupamiento*. El agrupamiento permite al juego enviar datos esenciales a los jugadores, utilizando el ancho de bando eficientemente. Permite además alta escalabilidad, para incluir un gran número de jugadores.

Lo primero a considerar es ¿por qué agrupar? Si se desea un juego que sea solo para dos jugadores (una pelea de boxeo), es necesario que uno sepa todo del otro, es decir, no tiene lógica agrupar. Pero si tendría sentido para uno en que hay una gran cantidad de jugadores en un escenario también grande, donde interactúan puntualmente solo grupos aislados de usuarios. Un ejemplo seria un juego de naves en el espacio exterior; no es necesario que un jugador reciba actualizaciones e información acerca de una batalla que ocurre fuera del radio de acción de su nave. Esto podría ser más abarcador, aún estado en el radio de acción, solo actualizar la información acerca del enemigo con el que se está enfrentando, y no de los demás, etc. El truco reside, en adicionar a *grupos* los jugadores que interactúan entre sí, o están en el rango de visibilidad, así no es necesario decidir a quién es necesario enviar datos cada vez que actualizamos a los demás; simplemente actualizamos a todos los integrantes de dicho grupo.

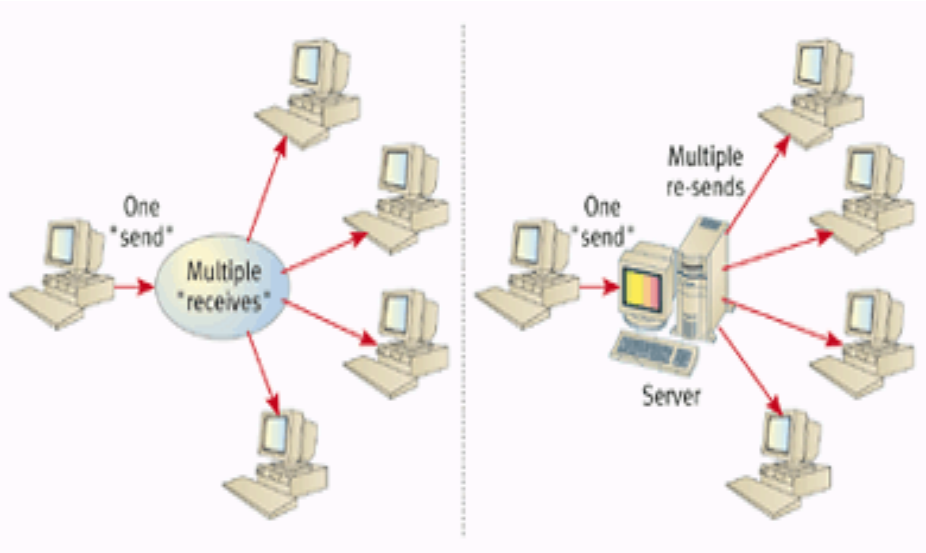


Figura 4: Transmisión multicasting vs. Explosión por Servidor.

El componente DirectPlay de DirectX, fue por mucho tiempo una de las mejores herramientas que proveían una capa de abstracción para transmisiones de uno a muchos y de muchos a uno, y una eficiente administración de grupos. La administración de grupos de DirectPlay provee la habilidad de definir, entrar, salir y enviar datos a grupos de jugadores. Es útil decir que DirectPlay, a pesar de ser muy conocida, está en desuso y está siendo cada vez más, sustituida por motores de red como RakNet, que es empleada por compañías tan prestigiosas como Sony Online Entertaiments y Codemasters.

Las tecnologías usadas no dicen cómo ni que tipos de grupo utilizar. De hecho, teóricamente en DirectPlay, cualquier jugador puede unirse a cualquier grupo. Esto no es muy óptimo ni lógico. Una forma de que funcione correctamente, es agrupando los paquetes según alguna medida que contengan, y enrutarlo hacia el grupo que esté basado en dicha medida. Existen variados métodos para sectorización, pero la agrupación por datos es la forma más intuitiva de crear los grupos, donde se extrae información del dato en sí, que podría ser localización geográfica del avatar, a que equipo pertenece la entidad, que tipo de entidad es, que aceleración posee, etc.

2.1.1- Agrupamiento por datos

Para muchos juegos, la sectorización geográfica es la vía más común. Bajo el sectorizado geográfico, el mundo virtual está dividido en diferentes regiones y los datos se agrupan según la región. Por ejemplo, en el esquema mostrado en Figura 5, se utilizan 12 grupos para segmentar el área de trabajo basada en una cuadricula de coordenadas rectangular Norte-Este. Cada entidad sabe a qué región pertenece y envía sus datos al grupo correspondiente. Así, el avión mostrado en la figura envía los datos al grupo 6. Es importante a notar que las entidades no tienen que confirmar su situación en la región con cada actualización. El agrupamiento facilita mucho el trabajo con técnicas predictivas (como Delta Reckoning, descrita en este mismo capítulo). Si se conoce que una región es de 100 millas o 20 años luz y cuán rápido el vehículo puede viajar, entonces se podría obtener la posición aproximada en que se encuentra o inclusive si se halla en otra región y así se puede posponer o adelantar, según sea necesario, la comprobación de una región basándose en esta predicción.

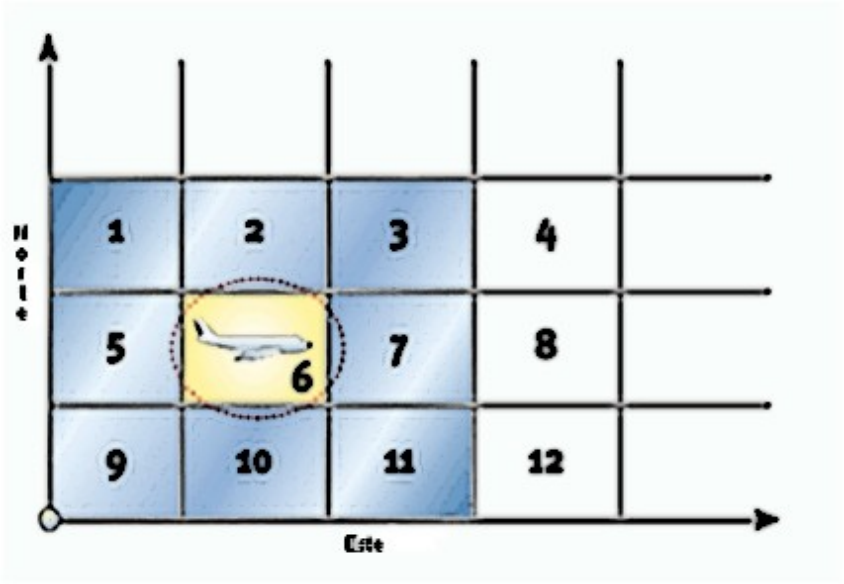


Figura 5: Agrupamiento geográfico.

La computadora de un jugador controlaría el flujo de datos que recibe, aceptando solo aquello que le interesa.

Por ejemplo, como se muestra en la Figura 2, una entidad en la Sector 6 sólo debería preocuparse de las entidades en su área, en este caso solo "escucharía" a las que se encuentran en el Sector 6. Si la entidad tiene una mayor área de interés, podría subscribirse a los grupos adyacentes, en ese caso agrega todos los grupos mostrado en azul.

El sectorizado geográfico es simplemente un ejemplo de agrupamiento por datos. En la práctica, cualquiera de los atributos de un objeto puede usarse para asignar ese objeto a una agrupación. Por ejemplo, un juego de guerra espacial podría ordenar los datos en dos grupos: los datos para las naves espaciales "invisibles" y datos para las naves del "visibles". Jugadores que no tienen la habilidad de ver las naves invisibles sólo escuchan al grupo de las visibles y viceversa, por consiguiente no gastan tiempo y ancho de banda en procesar los paquetes de las naves que pueden ver o no, etc.

2.1.2- Agrupación dinámica

El esquema de agrupación descrita previamente está basado en la definición estática de grupos. En el esquema del sectorizado geográfico, los límites de Sector 1, y por tanto, la definición del Grupo 1, se crean antes de que el juego inicie. Ésta es la manera más fácil de implementar grupos, partiendo desde la definición de grupo, y de las elecciones sobre dónde enviar o escuchar, puede codificarse junto con el juego.

Sin embargo, a veces las definiciones estáticas de grupo no funcionan. Se puede ejemplificar con la siguiente situación: En un juego de naves espaciales, una acalorada y decisiva batalla está teniendo lugar en Sector 6. El resto de espacio está relativamente vacío y prácticamente toda la actividad esta circunscrita al Grupo 6. En este caso, utilizando los grupos de la forma descrita anteriormente no mejora de manera apreciable el rendimiento de la simulación, ya que todos los jugadores envían y reciben al mismo tiempo. La solución en este caso sería la definición de grupos dinámicos. El servidor de grupo supervisa o infiere el comportamiento del flujo de tráfico entre los diferentes grupos y puede redefinir los grupos dinámicamente. En el juego de la nave espacial, por ejemplo, el servidor de grupo podría decidir dividir al Sector 6 en cuatro sectores más pequeños. Podría crea los grupos nuevamente, enumerándolos nuevamente según su nueva sectorización, y enviándoselas luego a cada jugador. También es posible redefinir los grupos existentes (por ejemplo, combinando los Sectores 4 y 8) para liberar grupos. La Figura 6 muestra un ejemplo de sectorización ajustada dinámicamente.

La línea de fondo es que con los grupos dinámicos, el software de cada jugador debe agruparse según la nueva definición transmitida por el servidor de grupo y modificar su estado de acuerdo a la nueva información.

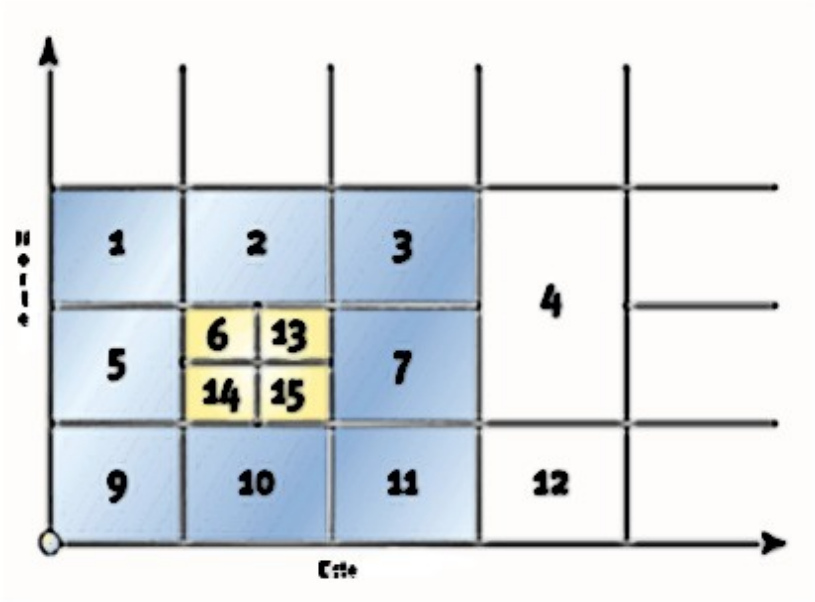


Figura 6: Sectorización dinámica de grupos geográficos.

Algo importante a tener en cuenta es que las entradas/salidas de grupos tardan. En general, la creación, unión de dos o más y eliminación de un grupo no ocurre instantáneamente. Por tanto, debe tenerse en cuenta a la hora de diseñar un juego la distribución de los datos y de la necesidad de disponer de los mismos a tiempo (quizás de antemano), para permitir que las operaciones con grupos se realicen con fluidez.

2.2- Relevancia

Un nivel o escenario de un juego podría ser grande, y en algún momento en concreto, un jugador puede ver solo una fracción pequeña de los actores en ese nivel solamente. La mayoría de los demás actores en el nivel no son visibles, no son audibles y no tienen ninguna interacción con el jugador. El conjunto de actores que un servidor determina que son visibles o capaces de afectar a un cliente, se denomina conjunto de actores relevantes para ese cliente. Una optimización de la utilización del ancho

de banda importante lo constituye el hecho de que el servidor solamente informa a cada cliente sobre clientes que pertenezcan a su conjunto de actores relevantes.

Unreal (5) aplica las siguientes reglas para determinar el conjunto de actores relevante de un jugador:

El actor es miembro(s) de la clase de ZoneInfo, entonces es relevante.

Si el actor tiene static=true o bNoDelete=true, entonces es relevante.

Si el actor es poseído por el jugador (Owner==Player), entonces es relevante.

Si el objeto es un arma y es poseída por un actor visible, entonces es relevante.

Si el actor está escondido (bHidden=true) y no colinda (bBlockPlayers=false) y no tiene un sonido ambiental (AmbientSound==None) entonces el actor no es relevante.

Si el actor es visible de acuerdo con su línea de visión entre la ubicación del actor y la ubicación del jugador, entonces es relevante.

Si el actor fue visible, por lo menos realiza 2 o 10 segundos (el número exacto varía debido a algunas optimizaciones de rendimiento), entonces es relevante.

Estas reglas son diseñadas dar una buena aproximación del juego de actores que puede afectar a un jugador realmente. Por supuesto, es imperfecto: la verificación de la línea de visión puede dar un falso negativo con actores muy grandes.

Las técnicas de verificación descrita con anterioridad podrían no ser óptimas para algún juego en particular y se podría recurrir a definiciones más cercanas a las características específicas de cada juego.

2.2.1 -El nivel de los detalles

Una vía de obtención de la relevancia de un objeto es mediante su nivel de detalles. Por ejemplo, de un objeto muy lejano como un avión, todo lo que una persona que camina por una calle puede ver es un punto en el cielo. Incluso si el avión estuviera volando rápido, solamente aparecería como un punto de lento movimiento. Si fuera el caso de una simulación o un juego virtual, la frecuencia de la sincronización y el nivel de detalle del modelo podrían ser tratados a más baja frecuencia si el objeto estuviera perceptiblemente más lejos del telespectador.

Los objetos espaciales pueden ser muy complicados y pueden estar conformados por cientos de millones de polígonos. Un modelo muy básico de un bombardero de B-25 elaborado con 3DsMax tiene más de 65,000 polígonos(6) .Representar y manipular uno de estos objetos o peor, muchos de ellos, es computacionalmente costoso. Sin embargo, pudiera no ser necesario tal nivel de detalle cuando el objeto es distante en la simulación. Diferentes niveles de detalles pueden ser definidos para los mismos objetos espaciales para distancias y orientaciones diferentes.

Técnicas de síntesis automática de modelos con menor nivel detalle en dependencia de la lejanía, pueden ser aplicadas a la zona de del objeto mostrado en el entorno virtual para reducir los requisitos computacionales y de transmisión de datos por la red.

En el contexto de ambientes virtuales distribuidos, el concepto del nivel de los detalles puede ser aplicado también a las sincronizaciones y predicciones. Objetivos lejanos podrían ser sincronizados menos frecuentemente, ya que sus estados son menos relevantes para el usuario

Podíamos definir una función propia de la aplicación para relacionar la frecuencia de actualización con la distancia a la que se encuentra el objeto a la que se le aplicara

La Figura 7 ilustra el concepto de nivel de detalle con algo más de claridad.

Figura 7: Tres tanques vistos desde tres escalas diferentes. En la distancia es difícil saber de qué tipo es, y actualizaciones frecuentes acerca de sus estados no son necesarias.

2.3- Priorización de actores.

En los juegos virtuales, es casi imposible mantener actualizado cada cliente sobre todos los eventos que ocurren, al mismo tiempo. Las limitaciones de ancho de banda no lo permiten, sobre todo en Internet. Asignar prioridades es algo que se utiliza mucho en los sistemas operativos, un proceso con mayor prioridad será atendido por el procesador con mayor presteza que uno de menor, lo que implica que se ejecutara una mayor cantidad de veces el de mayor prioridad. De manera análoga se le puede asigna una prioridad de actualización, a cada objeto que sea sensible a variar su estado en el juego.

Podría efectuarse asignando una variable denominada "Prioridad de red" o algo similar. Mientras más alta sea su "Prioridad de red", más ancho de banda que actor recibirá en comparación con otros, y mayor número de veces sus actualizaciones de estado serán enviados a los demás clientes. Un actor con una prioridad digamos, de 2.0, seria actualizado con el doble de frecuencia de un actor con prioridad de 1.0.

Es importante notar que es importante balancear las prioridades, no se consigue, por ejemplo aumentar el rendimiento de la red aumentando la prioridad de todos los objetos al máximo, sino todo lo contrario.

Un ejemplo clásico seria la manera que Unreal(5) realiza la priorización:

Avatares: 8.0

Bots: 7.0

Proyectiles: 6.0

Peones: 4.0

Criaturas decorativas (como peces): 2.0

Adornos: 0.5

2.4- Extrapolación, Dead Reckoning

2.4.1- El Dead Reckoning

Método usado en juegos y simulaciones en red para reducir la percepción de atraso causado por los problemas de latencia y ancho de banda. Los programas realizan esto prediciendo al estado futuro de una entidad basada en su estado actual (Como predecir el camino de un avión de combate basado en su velocidad y posición). Cuando un vehículo o entidad es creado, el nodo que posee la entidad envía un protocolo de estado de la unidad (PDU por sus siglas en ingles), al servidor, y de este se redistribuye a los demás nodos en la red. Este PDU contiene información que describe exclusivamente a dicho objeto, información sobre su estado cinético, que incluye posición, velocidad, aceleración, orientación, etc.

Por último, pudiera contener algún tipo de identificador, que dice a los demás nodos en la red que algoritmo específico de Dead Reckoning utilizar para dicha entidad. Cuando los restantes nodos reciben este PDU, crean copias locales del mismo. Todos los nodos comienzan a ver la entidad. El estado de dicha entidad es verificado cada cierto tiempo. El servidor envía correcciones acerca del estado actual de la entidad solo si esta no es lo suficientemente cercana al estado predicho.

También se utilizan algoritmos de predicción para rellenar los saltos y pérdidas entre actualizaciones de las entidades.

2.4.2- Delta Reckoning:

Es la más básica y clásica de las técnicas predictivas. Se basa en determinar la futura posición basándonos en la dirección, velocidad y quizás aceleración del mismo. Mientras el objeto se comporte del mismo modo, no serán necesarias actualizaciones, ganando en tiempo de respuesta y ancho de banda. Si un objeto ha recorrido 5 km hacia adelante en 5 min a velocidad constante, es presumible que para 10 min este a 10 km hacia delante, y así será si actualizaciones correctivas no informan de lo contrario. Es también posible que en muchos de los casos, las actualizaciones solo afecten a alguno de los componentes del movimiento del objeto en movimiento (dirección, sentido, velocidad, aceleración), pudiendo ser las correcciones menos complicadas. La Figura ejemplifica el uso de Delta Reckoning para un juego de aviones.

Cuando se inicia la simulación, el nodo creador del objeto informa de su estado y localización (vía servidor). En ese preciso momento el estado del avión esta sincronizado en cada nodo. A partir de ese instante, el avión se mueve a través de una ruta mostrada por el trazo de la línea continua. El avión puede maniobrar fuera de la de la línea fija, pero las actualizaciones corregirán el rumbo que toma el avión y luego se continuara aplicando Delta Reckoning para calcular su nueva posición. El trazo original continuo es el recorrido que realiza el dueño del avión. La línea discontinua es el recorrido del avión para los demás nodos después del Delta Reckoning.

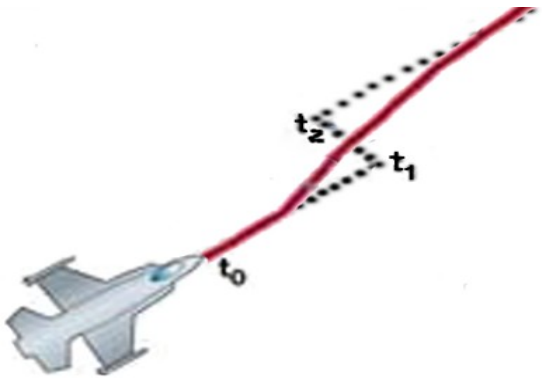


Figura 8 La trayectoria real se muestra de forma continua, la línea punteada muestra la trayectoria que predicha por el resto de los clientes, corregida a intervalos t.

Existen sub-variantes de este algoritmo, que corresponden a diferentes formulas físicas de estimación de distancia, aunque lo que cambia en todos los casos son los datos que se manejan, no la síntesis del algoritmo.

De Orden 0(mayor simplicidad) $x = x_0$, no ocurre movimiento, el objeto permanece en el mismo lugar,

Primer Orden (velocidad $x = x_0 + t_f * v$, ocurre movimiento a velocidad constante.

```
Para cada actualización
{
objecto.x_velocidad=object.x_DistanciaRecorrida/object.x_TiempoRecorrido;
objecto.y_velocidad=object.y_DistanciaRecorrida/object.y_TiempoRecorrido;
objecto.z_velocidad=object.z_DistanciaRecorrida/object.z_TiempoRecorrido;
}

Para cada frame
{
objecto.x_posicion = objecto.x_posicion + objecto.velocidadX;
objecto.y_posicion = objecto.y_posicion + objecto.velocidadY;
objecto.z_posicion = objecto.z_posicion + objecto.velocidadZ;
}
```

Segundo Orden (Aceleración) $x = x_0 + v_0 t + \frac{1}{2} a t^2$. Ocurre movimiento a aceleración constante.

```
Para cada actualización
{objecto.x_velocidad=(object.x_aceleracion * object.x_TiempoMovimiento  *
object.x_TiempoMovimiento)/2;
objecto.y_velocidad=(object.y_aceleracion * object.y_TiempoMovimiento *
object.y_TiempoMovimiento)/2;
objecto.z_velocidad=(object.z_aceleracion * object.z_TiempoMovimiento *
object.z_TiempoMovimiento)/2;
}
para cada frame
{
objecto.x_posicion = objecto.x_posicion + objecto.velocidadX;
objecto.y_posicion = objecto.y_posicion + objecto.velocidadY;
objecto.z_posicion = objecto.z_posicion + objecto.velocidadZ;
}
```

2.4.3- Target Reckoning

Esta técnica se aplica mayormente a los objetos o avatares, de los cuales, desde su posición actual (x_0, y_0, z_0), se sabe que para una variación de tiempo (Δt) conocida, se debe encontrar en una posición conocida (x_1, y_1, z_1). El servidor, se limita a informar a cada cliente del objeto en movimiento hacia qué posición se dirige, y en qué tiempo se debe encontrar allí. Cada cliente en que recibe esta información, mediante una función cualquiera, debe conseguir llevar correctamente el objeto hacia su futura localización en el tiempo correcto.

Una variante podría ser situar una serie de puntos entre punto de partida y objetivo, y luego mediante una función interpolante, trazar una curva o recta (en dependencia de la función empleada), que constituiría el camino que los une. Esto evita problemas como el del jugador inmortal y desfases en el momento de llegada de un avatar a un punto para clientes cualquieras.

Se añade un ejemplo en seudocódigo:

```
para cada frame
{
 objectivo.x_posicion = objectivo.x_posicion + objectivo.velocidadX;
 objectivo.y_posicion = objectivo.y_posicion + objectivo.velocidadY;
 objectivo.z_posicion = objectivo.z_posicion + objectivo.velocidadZ;

 proyectil.x = (proyectil.x + objectivo.x_posicion) / 2;
 proyectil.y = (proyectil.y + objectivo.y_posicion) / 2;
 proyectil.z = (proyectil.z + objectivo.z_posicion) / 2;
}
```

2.4.4- Statistical Move

Es una mejora al algoritmo inicial de Dead Reckoning. Se basa en “adivinar” el próximo movimiento, posición y/o jugada del avatar. ¿Cómo? Estadísticamente, según patrones predefinidos de las conductas más comunes que podrían seguir los jugadores, (aunque realmente seria efectivo con jugadores no humanos) y solo actualizando esta información el servidor cuando esta dista bastante de la predicha. Es decir, si un jugador cada vez que alguien se le acerca a menos de 20 metros, le realiza un disparo o varios en dependencia del arma, es presumible que ese será su comportamiento en el futuro. Si en cada nodo se realiza la misma estimación, probablemente los jugadores no humanos se

comportaran idénticamente, sin necesidad de actualizar su estado, sino muy de vez en vez. Para jugadores humanos crear patrones es algo más complicado, pero existen muchos comportamiento que son similares. Un jugador podría tener a alguien muy cerca, pero no desear disparar, pero casi es evidente que si un jugador le quedan pocas vidas es predecible que si tiene una vida relativamente cerca, la tomara, etc. Aparte de crear patrones de comportamiento predefinidos (sin lógica alguna para jugadores humanos), es posible crear patrones estadísticos empleando redes neuronales, más poderosas que un ser humano, pero con el inconveniente de que consumen muchos más recursos computacionales.

2.4.5- Server Reckoning

Se utiliza mucho en los juegos de estrategia (RTS por sus siglas en ingles), aunque pudiera resultar efectivo también en los de acción en primera persona.

Antes de que esta técnica surgiera, la única solución posible para garantizar la consistencia del juego era una técnica conocida como *frame-locking* (7)*,* donde cada nodo envía un mensaje de actualización por cada cuadro. Estas actualizaciones son enviadas a los demás clientes (en una configuración punto a punto) o hacia un servidor central que procesa el mensaje y reenvía el resultado a los demás nodos. Típicamente, lo que realmente ocurre es que se envían las actualizaciones a un intervalo fijo de tiempo, y no en cada cuadro. Esto hasta cierto punto podría resultar jugando en LAN, no así en internet, donde aún las latencias menores (150-300ms) serian catastróficas.

Es obvio que no todos las actualizaciones alcanzaran el servidor, producto de distintos factores, además de la latencia. Si alguna de las actualizaciones se pierde o demorara en el camino, el juego se congela, esperando la llegada de la actualización.

Si no se esperara por el nodo retrasado, se desincroniza totalmente la simulación. Solo el jugador retrasado ve lo que hizo, nadie más.

La figura siguiente ejemplifica este hecho mediante una línea de tiempo. En la primera actualización, el nodo A y el B envían sus actualizaciones al servidor. El servidor las recibe correctamente, casi al mismo tiempo, y reenvía el resultado inmediatamente. Sin embargo, para la segunda actualización, la

actualización se demora más en su tránsito, producto de la latencia. El servidor espera, y también el nodo A.

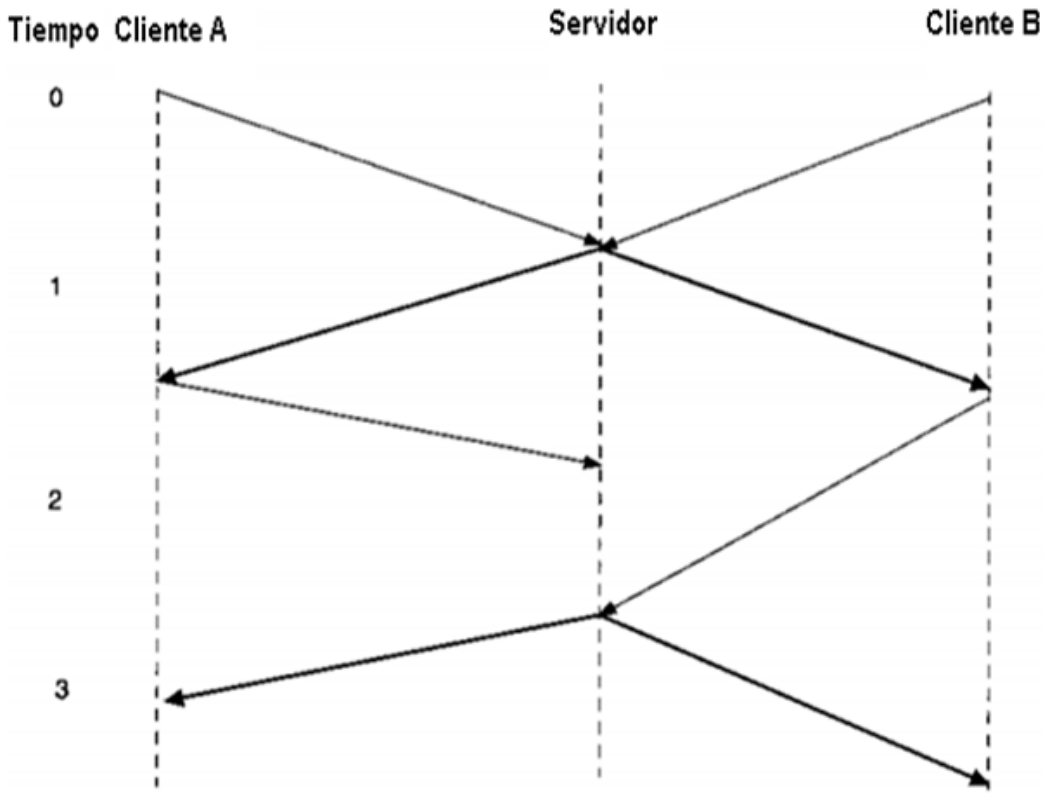


Figura 9: Congelamiento causado por el frame-locking. El servidor y el cliente A son retrasados por el cliente B.

La solución planteada por este algoritmo es, con los datos de que dispone el usuario, y en el caso de realización de movimiento, utilizando búsqueda de caminos*;* de forma similar a la que lo haría el servidor, comprobar si la acción o movimiento que pretende realizar el usuario es permisible o legal. Si es legal, realiza la acción y *después* de iniciada la misma, pide al servidor permiso para realizar la acción. Mientras el servidor realiza su propio chequeo autoritario de legalidad y tras comprobar, informa a los demás nodos del movimiento ejecutado, el nodo ya ha ejecutado el movimiento, sin congelarse esperando que se le permita ejecutarlo.

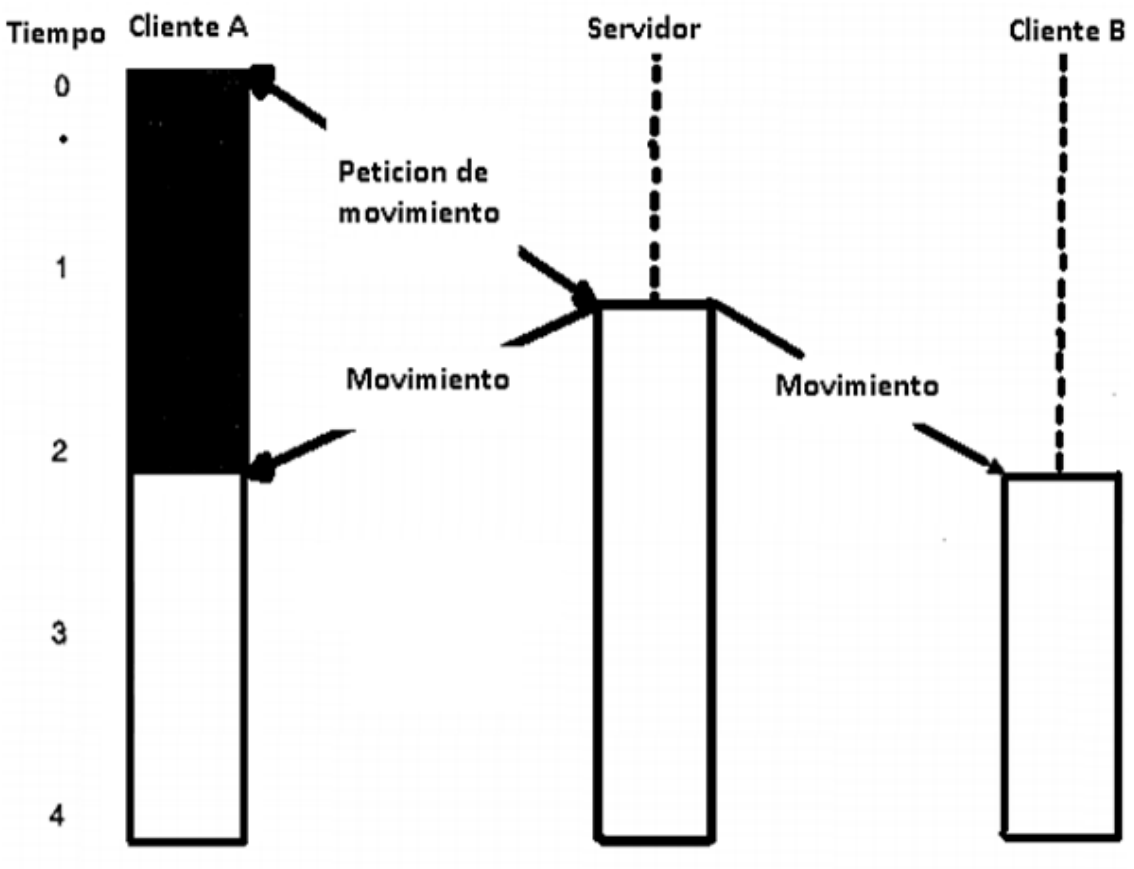


Figura 10: El jugador realiza el movimiento de forma "no oficial", si el servidor no rechaza el movimiento, este se convierte en "oficial", y el juego gana en tiempo de respuesta.

Esta técnica no es perfecta, presenta el inconveniente de desconocer las acciones que están realizando los otros nodos sobre el terreno que pretende recorrer. Ej. El camino que recorre el nodo A incluye un puente, pero el mismo fue destruido por el nodo B instantes después de iniciado el movimiento. El servidor autoritario debe realizarse cargo, entonces, de enviar una corrección al nodo (un nuevo camino quizás), que deberá corregir el anterior invalidado.

El algoritmo principal debería ser definido por tres métodos, de la siguiente forma:

-Hallar un camino temporal donde se ejecutara el movimiento e informárselo al servidor.

Se activa un camino temporal en el nodo cliente, que comenzaría a ser recorrido por el objeto. El camino en uso es hallado por una función generadora de camino, la cual a su vez ha recibido el inicio y el destino.

El camino tomado podría coincidir con el camino que encontrara el servidor, pero no hay garantías de ello, por tanto se envía, luego de iniciado el movimiento, una petición de confirmación de validez, adjuntando el principio y fin utilizados.

-El servidor realiza las verificaciones pertinentes.

El servidor recibe la petición de confirmación, y el basado en los datos de principio y fin de camino que recibe, realiza su propia búsqueda (el servidor posee más datos sobre el escenario que un nodo) verificando que el camino sea válido. El servidor halla el camino, con los mismos algoritmos (quizás búsqueda de caminos) que el nodo remitente de la petición, si el camino es válido deberían coincidir los resultados, de lo contrario, el servidor realiza valer su autoridad, y le envía al nodo remitente de la petición el camino correcto.

-Corregir un camino incorrecto.

Esta función solo seria invocada si es recibido un nuevo camino por parte del servidor autoritario. Este es el inconveniente principal de la técnica. Si el camino no coincide debe, interpolando, realiza converger su posición actual con la nueva posición inicial y recorrer solo entonces el nuevo camino.

2.4.6- Path Reckoning:

Se utiliza mucho en los juegos de estrategia (RTS por sus siglas en ingles), y para las entidades no controladas por jugadores (NPC, por sus siglas en ingles). Es muy útil para predecir la posición de los NPC, ya que estos tienen un movimiento limitado a un camino fijo por lo general. También para conocer la posición de un objeto del que se conoce el principio y fin del camino, que está recorriendo, como en los RTS. Un objeto (digamos, un hombre a caballo), se le ordena que valla de una posición a

otra. Si su movimiento es a velocidad constante basta, en la mayoría de los casos con conocer el principio y el fin del camino para conocer su posición. Podría dar resultados interesantes en unión del Server Reckoning en un RTS. En juegos de acción en primera persona resulta útil mientras el jugador se mueva por un camino predecible, que es el caso de los NPC.

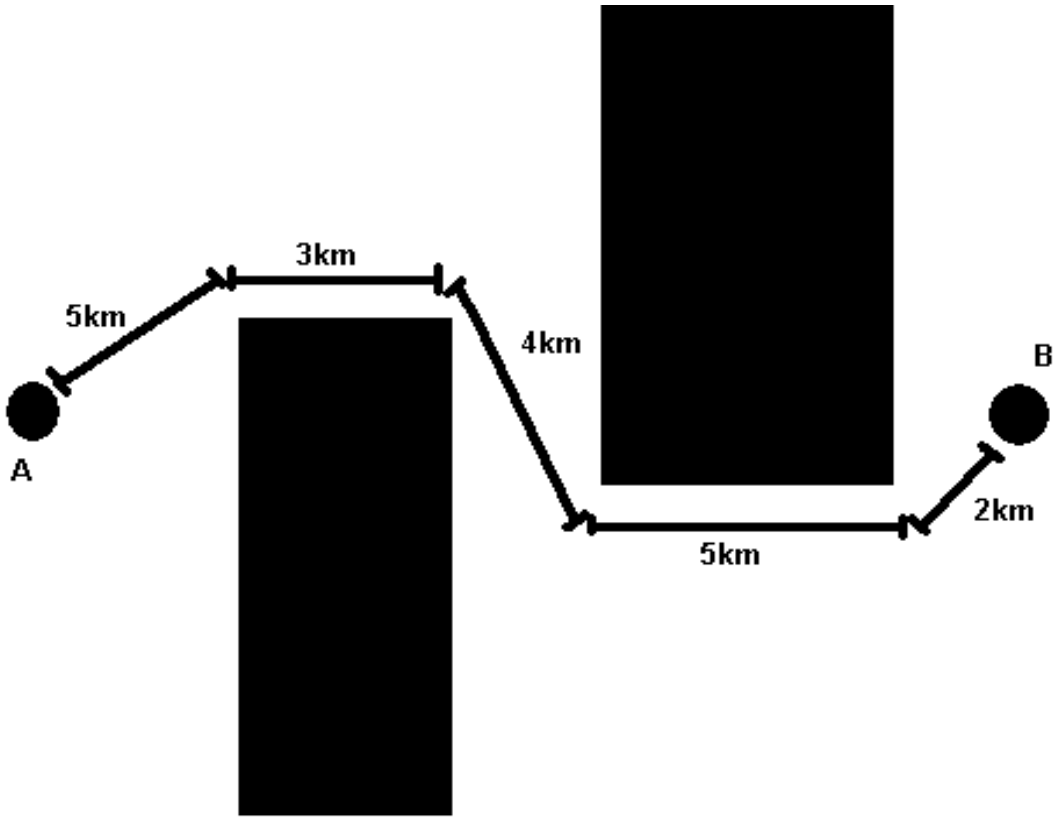


Figura 11. Conocido el camino y la distancia a recorrer es sencillo conoce su posición para un tiempo dado.

-Hallar la posición en un camino dado:

Se activa un camino en el nodo cliente, que comenzaría a ser recorrido por el objeto. El camino en uso es hallado por una función generadora de camino, la cual a su vez ha recibido el inicio y el destino.

Se calcula la distancia total del camino (Posición final – posición inicial). Luego, conocida la velocidad del objeto que se mueve, es posible predecir en qué punto del camino se encuentra el objeto.

2.5 - Interpolación

2.5.1- Definición General

Se denomina interpolación a la construcción de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos. Es frecuente disponer de un cierto número de puntos obtenidos por muestreo o a partir de un experimento y pretender construir una función que los ajuste. Otro problema estrechamente ligado con el de la interpolación es la aproximación de una función complicada por una más simple.

Si tenemos una función cuyo cálculo resulta costoso, podemos partir de un cierto número de sus valores e interpolar dichos datos construyendo una función más simple. En general, por supuesto, no obtendremos los mismos valores evaluando la función obtenida que si evaluásemos la función original, si bien dependiendo de las características del problema y del método de interpolación usado la ganancia en eficiencia puede compensar el error cometido. En todo caso, se trata de, a partir de n parejas de puntos (xk, yk), obtener una función f que verifique su existencia, a la que se denomina función interpolante de dichos puntos.

$$f(x_k) = y_k \, , k = 1, \dots, n$$

A los puntos xk se les llama nodos.

2.5.2- Interpolación en juegos virtuales y simulaciones:

Es empleada para mover algún objeto o entidad desde alguna localización hacia otra gradualmente en el tiempo.

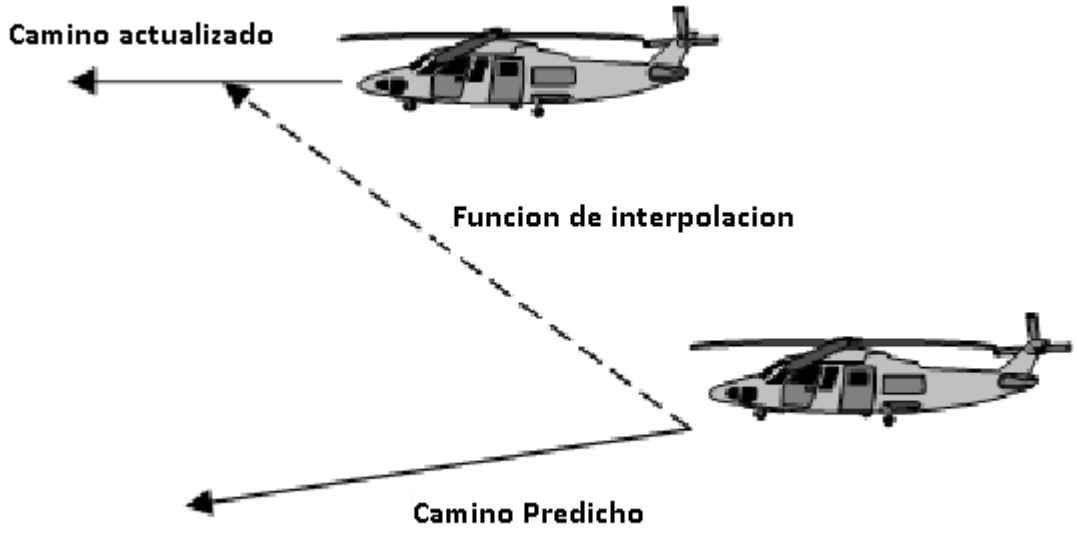


Figura 11: Una posición es corregida, no recolocando el objeto en su posición correcta, ya que le resta presencia al juego, sino mediante una función interpolante, haciendo converger el objeto hacia la posición correcta

Existen distintas variantes de técnicas, difiriendo muy poco en algunos casos. En este documento se explican cuatro de las más empleadas:

- Interpolación lineal, independiente de la frecuencia de imágenes por segundo.
- Final suave, utilizando cálculo de coma flotante.
- Final suave, dependiente de la frecuencia de imágenes por segundo, utilizando cálculo entero.
- Final y principio suaves, independiente de la frecuencia de imágenes por segundo

Todos estos métodos comparten una base común. Se inicia en un punto cualquiera, tratando de llegar hasta otro, pudiendo tener un límite o no, de tiempo para alcanzar su destino. El comienzo y el destino deben ser cualquier valor numérico o combinación de valores numéricos.

Los mismos pueden ser valores de temperatura, altitud, posición 3D, vector de velocidad o de dirección, o cualquier factor que pueda ser descrito numéricamente. La interpolación es simplemente

llevar un valor hacia otro a través de una ruta suavizada de diferentes valores intermedios. Si se pretende aplicar esta técnica a u valor vectorial, se le debe aplicar a cada componente por separado.

2.5.3- Interpolación lineal, independiente de la cantidad de imágenes por segundo.

En el caso de la interpolación lineal, se desea calcular una velocidad ideal al principio del movimiento y luego aplicar dicha velocidad a cada cuadro. Este método resulta en una ecuación que produce un trazado en línea recta, como el trazado en la Fig. 12. Con algunos conocimientos elementales de física se comprende rápidamente la funcionalidad de la ecuación 1.

$v = (x_f + x_0) / t$ (1)

Luego es necesario aplicar dicha velocidad a cada cuadro. Para realizar esto correctamente es necesario conocer cuánto tiempo toma cada cuadro .Obtenido este dato, es posible calcular el cambio de posición, utilizando la velocidad anteriormente calculada, y adicionarla al valor original,

$x = x_0 + t_f * v$ (2)

Se adiciona una clase de ejemplo, que se denomina CLinearInterpolation.

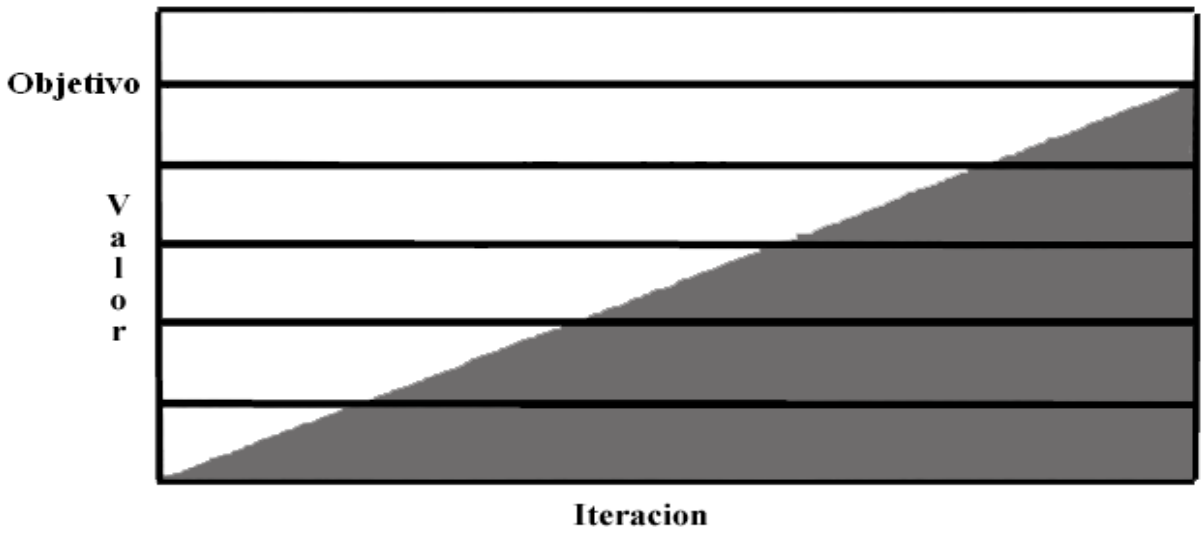


Figura 12: Interpolación lineal

```
class CLinearInterpolation
{

 float value;
 float step;
float remainingTiime;

public:

 bool Setup(float from, float to, float time)
 {
  if(time<0)
  {
  return false;
  }
 remainingTime=time;
 value=from;
 step=(to-from)/time; //Calcula distancia por segundo.
 return true;
 }

bool Interpolation(float deltaTime) //devuelve True si se alcanza o pasa el final.
{
 remainingTime -= deltaTime;
 value+=step*deltaTime;
 }

float GetValue()
{
 return value;
 }

};
```

2.5.4 -Final suave, dependiente de la cantidad de imágenes por segundo, utilizando cálculo de coma flotante.

Este método es dependiente del numero imágenes por segundo, como consecuencia de esto, se comportara diferente si es invocado a 10 imágenes por segundos, que si se invoca a 20. Este método es por tanto, aconsejable si la precisión no es demasiado importante.

El concepto detrás de este método es que se quiere calcular un promedio superior del valor inicial y llegar al valor deseado aumentando dicho promedio progresivamente mediante un factor de

incremento. El nuevo valor x se iguala a x_0(valor original), multiplicado por el factor de incremento, adicionado al valor final de destino x. La suma es dividida por el factor para preservar la escala. La x resultante es empleada como x_0 para la proxima iteración de la ecuación.

$$x = (x_0 * (factor - 1) + x_f)/factor \quad (3)$$

El factor es un valor mayor que 1 para logra el comportamiento esperado para la ecuación. A mayor valor del factor se consigue que sea mayor el tiempo que tome el algoritmo para alcanzar el valor final. Esto genera una curva suave, como se muestra en la figura, que muestra como el valor cambia rápidamente al principio, para luego acomodarse suavemente hacia su destino final, aproximándose a un *final suave.*

La clase de ejemplo en C++ que se adiciona como anexo, se denomina CEaseOutDivideInterpolation. Con esta implementación en coma flotante, puede tomar algo de tiempo alcanzar un estado estable.

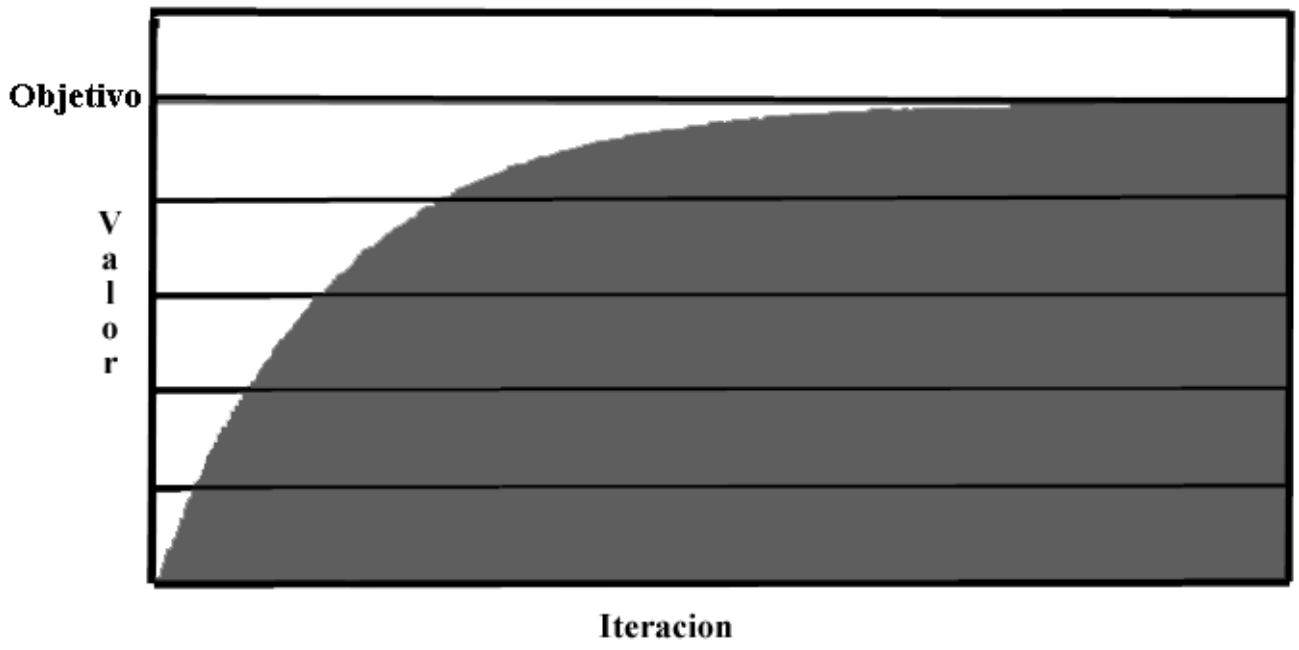


Figura 13: Final suave, coma flotante.

```
class CEaseOutdivideinterpolation
{

 float value;
 float target;
 float divisor;

 public:

 bool Setup(float from, float to, float divisor)//initialize los valores
 {
  if(divisor<=0)
 {
  return false
  }
 this->value=from;
 this->target=to;
 this->divisor=divisor;
 return true
 }


bool Interpolate() /* No dependiente del tiempo, llamado en cada iteración, devuelve True si no cambia el valor, que significa que se alcanzo el final */
{
 float v=value;
 if(divisor>0)
 {
  value=(value * (divisor- 0.0f) +target)/divisor
 }
 //No muy a menudo True preferentemente.
 return (value==v)
}

float GetValue() //llamado al final de la ejecución, devuelve el value necesario para la próxima iteración.
{
 return value;
}

}
```

2.5.5 -Final suave, dependiente de la cantidad de imágenes por segundo, utilizando cálculo entero

Este método al no utilizar división, es menos costoso computacionalmente. Funciona rápido, pero tiene restricciones mayores en cuanto a flexibilidad que el método anterior.

El proceso de incrementar el promedio utilizando, produce un efecto curioso. El proceso de cambio tiende a atascarse en determinados niveles durante la interpolación, y al finalizar cada iteración pudiera parece que no se alcanzara nunca el punto de destino. La ecuación muestra las modificaciones realizadas. Los valores de $(2^n\text{-}1)$ y n son evaluadas de forma tal que se garantice la rapidez del código. Dadas las dificultades al terminarse la iteración, la curva es mucho menos suave que la anterior exhibida por el método de coma flotante.

$$x = (x_0 * (2^n - 1) + x_f) \gg n \; ; \; Ej: x = (x_0 * 7 + x_f) \gg 3 \quad (4)$$

A pesar de este inconveniente, la reducción de tiempo de procesamiento puede ser importante cuando son muchos los valores.

Se adiciona una clase de ejemplo, que se denomina: CEaseOutShiftInterpolation.

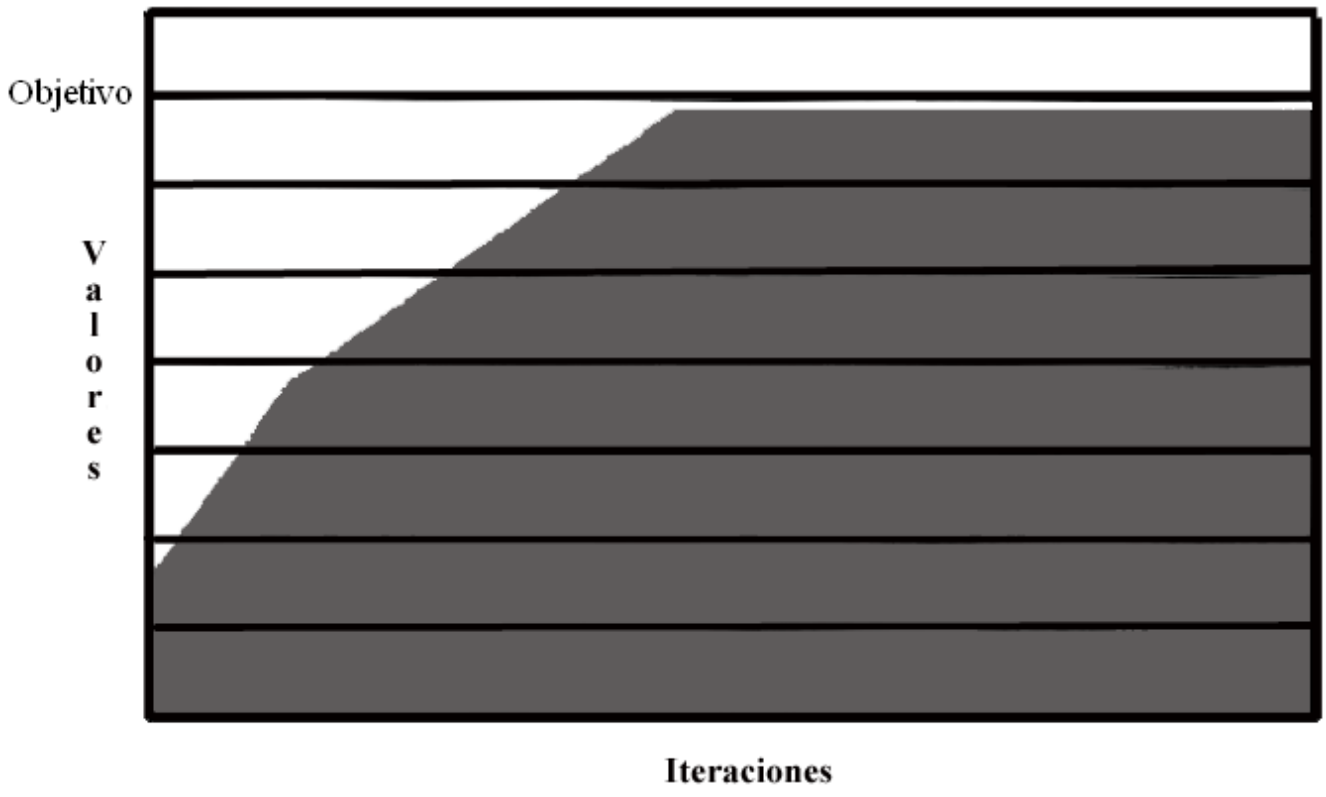


Figura 14: Final suave, utilizando cálculo entero.

```
class CEaseOutShiftInterpolation
{

 int value;
 int target;
 int shift;

 public:
 bool Setup(int from, int to, int shift) /*inicializa los valores, donde shift controla la rapidez de cambio de la interpolación */
 {
  if(shift<=0)
 {
 return false;
 }
 this->value=from;
 this->target=to;
 this->shift=shift;
 return true;
 }
 bool Interpolate() //No dependiente del tiempo, llamado en cada iteración, devuelve True si no cambia el valor, que significa que se alcanzo el final
 {
  int v=value;
  if(shift > 0)
  {
  value=(value*((1 << shift)-1)+target)>>shift;
  }
  return (value==v);
 }
 int GetValue() //llamado al final de la ejecución, devuelve el value necesario para la próxima iteración.
 {
 return value;
 }

};
```

2.5.6 -Final y principio suaves, independiente de la cantidad de imágenes por segundo

Esta técnica requiere conocimientos de física un poco mayores que la anterior, pero no mucho más. El objetivo es producir correctamente un principio y final suaves. Es necesario comenzar con una velocidad inicial igual a cero, aumentar la misma a aceleración constante hasta alguna velocidad en un

punto medio, para luego realizar el mismo proceso, pero inverso, hasta alcanzar el punto de destino con una velocidad igual a cero. Este proceder es ilustrado en la Fig. 15.

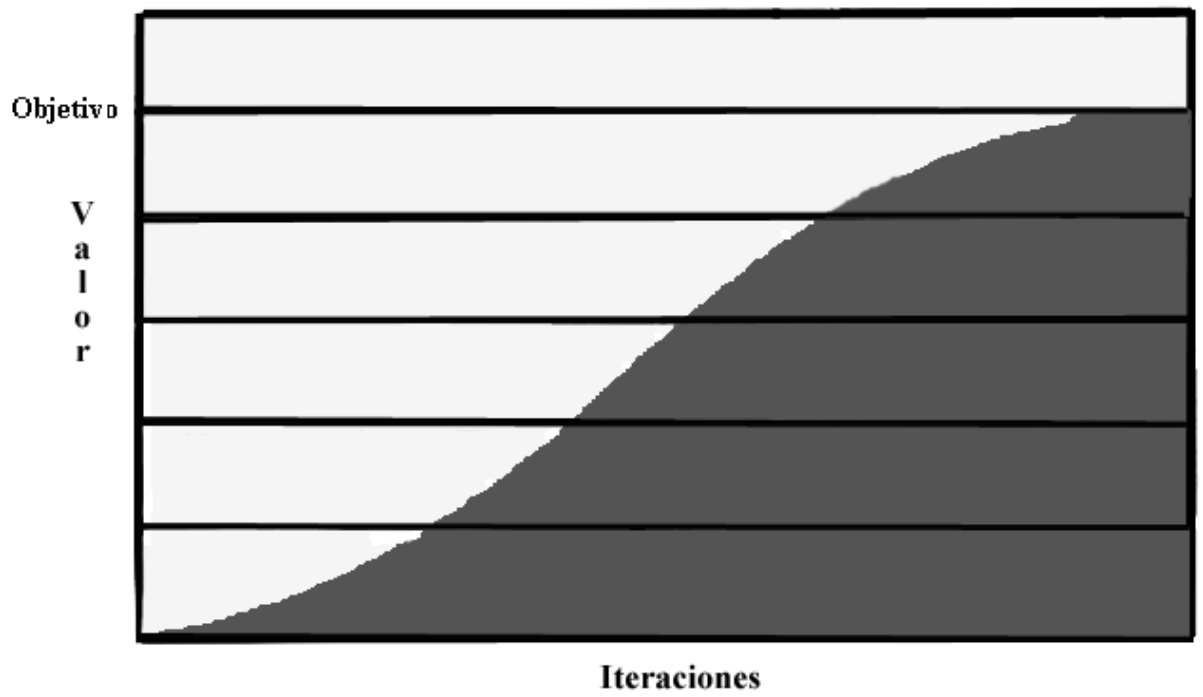


Figura 15: Final y principio suaves

El primer paso es calcular la aceleración requerida. Luego simplemente se invierten los valores para la segunda mitad (se realizarían las mismas operaciones, pero a la inversa). Aunque las formulas pueden ser encontradas en cualquier libro de física elemental, se ejemplificaran para una mayor comprensión. Se necesita comenzar con velocidad cero, con x representando el promedio entre la posición inicial y final, hasta alcanzar el destino final.

$x = x_0 + v_0 t + \frac{1}{2} a t^2$ (5). $a = \frac{2(x - x_0)}{t^2}$ (6).

Una vez que la aceleración es conocida, es necesario aplicar esa aceleración a cada cuadro. Este método es similar a la forma en que es realizada en la interpolación lineal, donde es necesario tener en cuenta el tiempo de cada cuadro.

Al comenzar la velocidad es cero. Para cada cuadro, es determinado si se está en la primera o la segunda mitad del camino, para así saber si acelerar o desacelerar. La velocidad es incrementada o disminuida en algún resultado de la ecuación (7), entonces la velocidad es aplicada a la posición, como se muestra en la ecuación (5), la misma que para la interpolación lineal, pero con una velocidad que cambia para cada cuadro. Esta velocidad debería converger cada vez más hacia cero, cuanto más cerca se acerca al destino, pero podría no ser exactamente cero, dado a posibles errores cometidos en las iteraciones.

$v = v_0 + at$ (7)

Se adiciona una clase de ejemplo, que se denomina CEaseInOutInterpolation.

```
class CEaseInOutInterpolation
{

 float value;
 float target;
 float remainingTime;
 float totalTime;
 float speed;
 float acceleration;

private:
 bool Setup(float from, float to, float time)
 {
  if(time<=0)
  {
 return false;
  }
 value=from;
 target=to;
 speed=0.0f; //derivado de x= x0 + v0^t + a*t/2
 acceleracion=(to-from)/(time*time/4);
 remainingTime=totalTime-time;
 return true;
 }
 bool Interpolate(float deltaTime)
 {
 remainingTime-=deltaTime;
 if(remaining<totalTime/2)
 {
 //Desaceleración
 speed-=aceleration*deltaTime;
```

```
}
else
{
//Aceleración
speed+=acceleration*deltaTime;
}
value+=speed*deltaTime;
return(remainingTime<=0);
}

float GetValue()
{
return value;
}

}
```

La interpolación puede ser muy útil, pero como casi todo, presenta inconvenientes y dificultades. Los ángulos son problemáticos. Podría resultar que alguno de los algoritmos aquí descritos asumiera que el ángulo medio entre 1° hasta 359° sea siempre 180°, cuando la respuesta correcta quizás sea 0°, dependiendo de cómo se desea el comportamiento de la interpolación. Es necesario asegurar que valores que pueden ser expresados de diferente forma (como los ángulos), estén en el rango correcto.

Pudiera ser aconsejable adaptar y acomodar los algoritmos a las necesidades puntuales, que podrían variar.

2.6 -Compresión:

La compresión de datos consiste en la reducción del volumen de información tratable (procesar, transmitir o grabar). En principio, con la compresión se pretende transportar la misma información, pero empleando la menor cantidad de espacio .La compresión se basa en las ideas de redundancia y entropía. Cuantos más datos redundantes y menos entropía hay, más los podemos comprimir. Es decir que lo ideal para lograr compresiones altas son archivos con muchos datos redundantes y con un cierto orden.

Así, por ejemplo, si en un fichero aparece una secuencia como "AAAAAA", ocupando 6 bytes se podría almacenar simplemente "6A" que ocupa solo 2 bytes, en algoritmo RLE. En realidad, el proceso es mucho más complejo, ya que raramente se consigue encontrar patrones de repetición tan exactos.

La compresión de mensajes puede ahorrar un importante ancho de banda en un sistema distribuido. La compresión y descompresión ocurren en la fuente y destino respectivamente y desgraciadamente, sólo pueden lograrse a costa de un costo computacional agregado en cada receptor y emisor. Sin embargo, puede ser vital su utilización en redes de un ancho de banda limitado, son múltiples e inmediatas las ventajas su implementación.

El proceso de compresión puede ser interno o externo. La compresión interna manipula los paquetes según su contenido y no en lo que se ha transmitido previamente, mientras que la externa manipula los paquetes basándose en lo que ha sido transmitido previa y eficazmente. La opción correcta para la elección del tipo de algoritmo de compresión depende de la aplicación en particular. La decisión depende de la frecuencia de actualización de paquetes, del volumen de los paquetes, de la arquitectura de comunicación y los protocolos usaron para la distribución de los paquetes.

2.6.1 -Compresión Delta:

Es una técnica empleada para la transmisión o almacenamiento de datos, en forma de diferencias entre una secuencia de datos y el archivo completo. En ambientes donde se utiliza para archivar historiales de cambio (Ej.: Proyectos de Software), se le conoce como Codificación Delta.

Las diferencias son guardadas en archivos discretos denominados "deltas". Dado que en muchos casos los cambios de los datos son pequeños, se reduce enormemente la redundancia en los datos y la transmisión de los datos comprimidos de esta forma utiliza de una forma mucho más eficiente el ancho de banda que sus equivalentes no comprimidos. Desde un punto de vista lógico, la diferencia entre valores idénticos(o al menos equivalentes), es conocida como 0 o neutro. Una buena Delta debe ser mínima o ambigua, mientras algún elemento del par no esté presente.

Una delta puede ser definida de dos formas diferentes, Simétrica y Directa. La Simétrica puede ser definida como:

$\Delta (v_1, v_2) = (v_1 \setminus v_2) \cup (v_2 \setminus v_1)$, donde v_1 y v_2 representan dos valores diferentes.

La Directa puede ser una secuencia de operaciones de cambios (elementales), que transforman el valor original v_1 en uno nuevo v_2.

En la siguiente imagen, podemos ver un ejemplo de implementación de este tipo de compresión para video. En la parte superior se puede observar una secuencia de imágenes completas (dividida por fotogramas), que pasadas a gran velocidad crean sensación de movimiento (el principio básico del cine). La mayoría de los códec actuales (XviD, DivX, H.264, etc.), recogen las partes de la secuencia que varían de un fotograma a otro: en la parte inferior de la imagen, los fotogramas llaves 1 y 5 son imágenes completas; que serían introducidas a intervalos continuos de tiempo. Por ejemplo, durante una conversación el fondo queda estático y lo único que varía son las caras de los personajes, por lo que no es necesario modificar el fondo y por ello no se actualiza su estado, ahorrando espacio. Las caras de los personajes sí se van actualizando, pues varían con el tiempo. Así una secuencia de imágenes es posible comprimirla, sin sacrificar apenas la calidad.

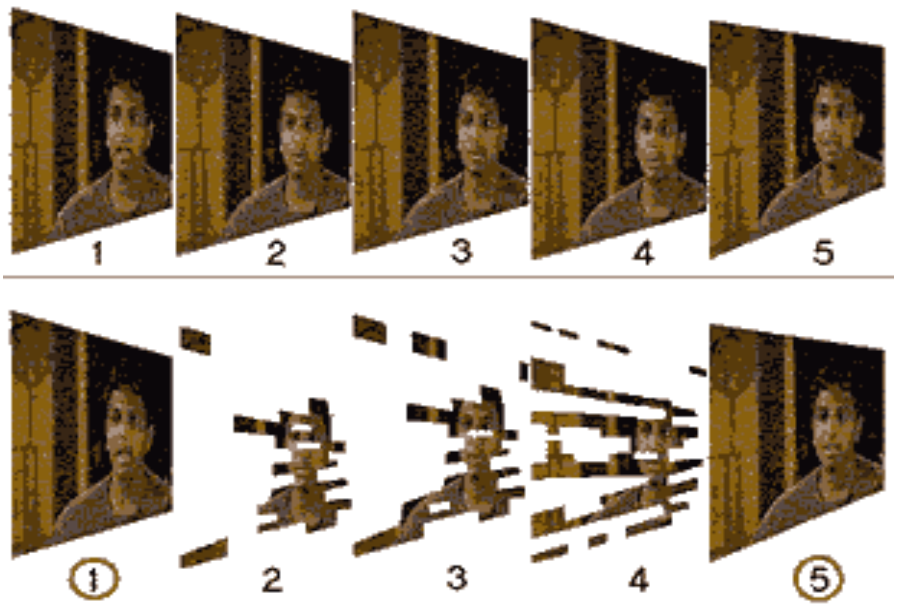


Figura 16: Compresión delta, empleada por códec de video.

Este tipo de compresión, en el caso de su aplicación para redes pudiera ser conseguida utilizando un bit (o cualquier tamaño de dato que se prefiera), para decir que incluye un mensaje que cambia un

estado. Este proceder es denominado abanderamiento, y cada bit incluido en cada segmento del paquete es una bandera. Tenemos el ejemplo de la siguiente figura:

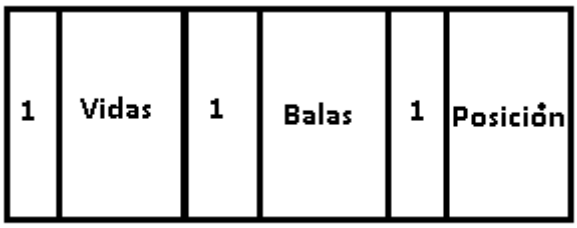


Figura 17: Paquete sin comprimir.

El paquete lleva información hacia el cliente acerca de diferentes variables que definen su estado (vidas, balas, variación de posición).

El otro caso sería el del ejemplo de la siguiente figura:

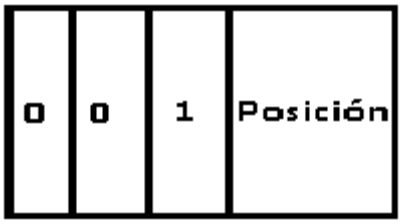


Figura 18: Paquete Delta-comprimido.

Este ejemplo significa que se mantiene constante la cantidad de vidas y de balas, y varía su posición. Obsérvese que solo se envía un cero en el encabezado de cada segmento, y no se envía el segmento como tal, solo el de cual estado varia, que contiene la actualización correspondiente.

2.7- Servidor autoritario

Las predicciones del cliente funcionan pronosticando de forma adelantada y utilizando formulas físicas y matemáticas localmente en el cliente, utilizando directamente la información de que dispone, sin esperar confirmaciones ni actualizaciones por parte del servidor. Sin embargo estas predicciones están

sujetas a errores y se hace necesario que el servidor actualice el estado del juego periódicamente. El estado del juego que posee el servidor es totalmente definido por variables obtenidas de todos los actores del juego que interactúan entre sí.

Que el servidor sea autoritario sobre el estado del juego implica que la visión que el posee del estado del juego es la correcta y el estado del juego que poseen los clientes debe ser consistente con la del servidor. Los clientes muy a menudo, al poseer menos datos que el servidor, poseen un estado del juego que es solo una aproximación al estado "real" del juego, que sería el estado del servidor. Los objetos que definen la simulación que ocurre en cada cliente deben ser considerados como una representación aproximada y temporal del objeto, y no el objeto en sí.

Siempre el servidor es autorizado sobre la simulación y las interacciones, así que incluso si el cliente intenta realizar trampa, todo lo que consigue es engañarse a sí mismo, ya que no puede alterar el estado del juego que posee el servidor. Visto en concreto, todas las comprobaciones y chequeos de legalidad ocurren en el servidor, y lo que él decida es lo que será.

Las complicaciones mayores debido a su implementación, ocurren en el lado del cliente, sobre todo cuando su predicción es errónea y necesita ser corregida. La corrección es realizada por el cliente, pero la orden de ejecución es competencia del servidor, que debería informar el valor real de la variable errónea (posición, cantidad de balas, armadura, etc.). El servidor es autoritario también en esferas como los servicios administrativos, estadísticas de marcas, etc.

2.8-Ejemplo de utilización de algunas técnicas en juegos comerciales:

2.8.1- Quake 3:

Antecedentes:

La primera y real implementación de de la que se tiene noticia fue en 1995, utilizando TCP para QuakeTest(QTest). Resulto suficiente para jugar en LAN, pero en Internet la perdida de paquetes y las latencias que esto implicaba, además de las inherentes a las conexiones, fueron lo suficientemente grandes para que se considerara utilizar UDP (Quake 2).El uso de UDP, tanto en su modo original no fiable, y su implementación fiable, fue la elección para iteraciones posteriores.

Su efectividad fue evidente y se considerara posteriormente como una norma de aplicación de red. Sin embargo, la aplicación de UDP fiable era algo novedoso, y se enfrentaba con muchos errores de implementación, además de pérdidas de secuenciación.

Quake promueve una arquitectura cliente-servidor monolítica. Un ordenador es designado "Servidor", y es responsable de realizar todas las decisiones pertinentes al juego. Los demás ordenadores son "Clientes", y fueron concebidos en un principio, solo como terminales de representación gráfica, que enviaban peticiones y recibían los resultados de estas peticiones hechas al servidor como una lista de objetos y eventos a representar.

Este progreso permitió el juego por Internet a gran escala, cuando servidores independientes surgieron por toda la red. La arquitectura cliente-servidor fue extendida por QuakeWorld y Quake 2 después, que cambió de lugar algunas simulaciones adicionales. Los algoritmos predictivos del lado del cliente (Dead Reckoning), en su variante más clásica Delta (en aquellos momentos algo muy novedoso), tienen su iniciación. Todo esto persigue disminuir el uso de ancho de banda, al disminuir la cantidad de información a ser transmitida.

El cliente recibe no sólo una lista de objetos para representar sino también la información sobre sus trayectorias así que el cliente puede realizar pronósticos rudimentarios acerca de los movimientos de los jugadores utilizados para eliminar las latencias percibidas en el movimiento del cliente.

Con la siguiente iteración (Quake 3), se dejo atrás la antigua noción de paquete fiable entero y compacto, sustituyéndolo con paquetes que contenían solamente los datos necesarios, omitiendo datos redundantes. Causo un gran impacto la utilización pionera del envío de paquetes de actualizaciones delta comprimidos del servidor hacia el cliente. Los paquetes comprimidos mediante este método se enviaban de manera secuenciada. El servidor no espera la confirmación de recepción de un paquete, solamente envía actualizaciones cada cierto intervalo de tiempo. Además de la compresión Delta, se comprime cada paquete mediante el algoritmo Huffman.

Desventajas:

Desincronización de las predicciones: en este modelo, el código de implementación de la red y el código del juego como tal se encuentran separados en módulos distinto, por tanto cada uno debe estar constantemente sincronizando con el otro, pudiéndose perder algo de tiempo en mantener dicha

sincronización; pero por otro lado, la concatenación estrecha entre módulos distintos realiza la extensibilidad difícil.

A pesar de haber sido un juego que sentó muchas pautas, sigue consumiendo demasiado ancho de banda por una ineficiente implementación de la capa de fiabilidad de red.

2.8.2- Unreal Tournament

Introduce al juego en red un nuevo concepto de modelo cliente-servidor más generalizado. De este modo, el servidor es aún autoritario acerca de le estado del juego y su evolución, sin embargo, el cliente implementa localmente las comprobaciones y predicciones del mismo modo que lo haría el servidor. De esta manera se consigue, que aunque el servidor posee más datos acerca del estado del escenario del juego, las predicciones del cliente puedan ser bastante acertadas.

Introduce el concepto de relevancia y priorización, consiguiéndose que los datos realmente importantes lleguen a tiempo para que la simulación fluya correctamente. Prioriza objetos que son relevantes para el jugador, que podrían ser los enemigos cercanos visible, muro que se ha destruido y le impide el paso, etc. Un enemigo oculto no es relevante, mientras no se muestre, no es relevante un enemigo que este en otra sala, etc. Le asigna prioridades a los objetos de la simulación, y serán enviados con mayor frecuencia actualizaciones del estado de aquellos objetos que sea más importantes.

También realiza manejo de roles; esto significa que cada avatar u objeto tiene un "dueño". El dueño del avatar puede ser el servidor, el propio cliente u otro cliente. Si el "dueño" de objeto es del servidor, el cliente espera a que el servidor sea el que actualice el estado del mismo, de lo contrario, sobre los de otros clientes debe efectuar predicciones. Sobre los que les pertenece, actúa directamente. Es muy útil para que el cliente interactúe solo en su área de competencia.

Realiza compresiones de red y predicciones similares a las de Quake, con algunas diferencias especificas en el delta, ya que utiliza variables diferentes (Quake: Position += PositionIncrment Unreal:Position += Velocity *DeltaTime). De esta manera se logra una independencia de la cantidad de imágenes por segundo, aunque sean al fin al cabo las dos formas equivalentes en teoría.

Capitulo 3: Demo de algunas de las Técnicas.

3.1- Modelo de Dominio:

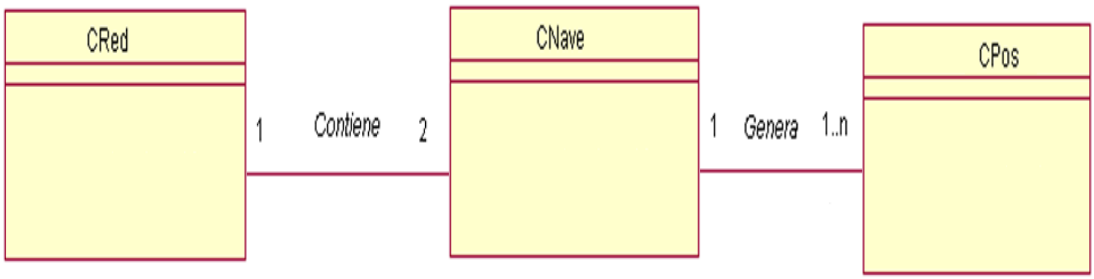


Figura 19: Modelo de dominio.

3.1.1- Especificaciones del dominio

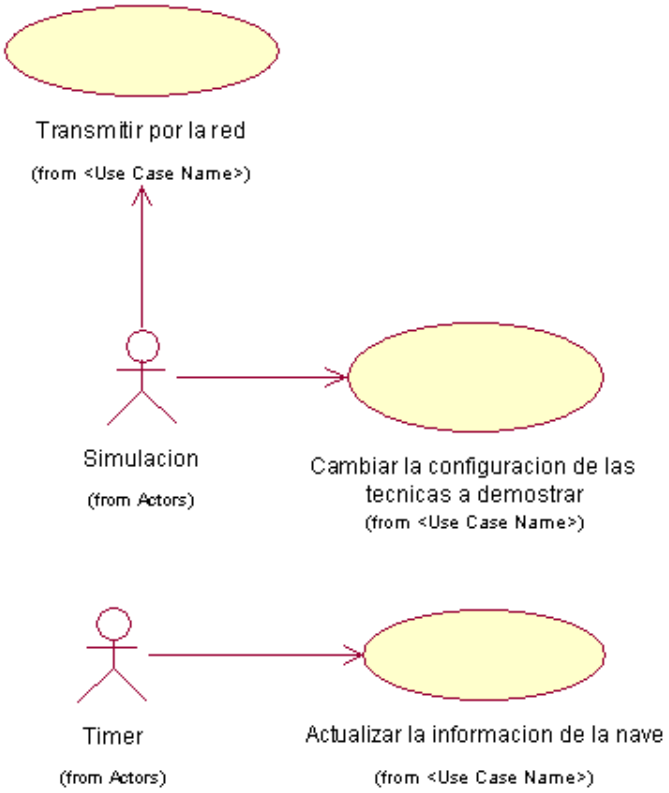
A continuación se presentan un conjunto de conceptos que conforman el glosario de términos del modelo del dominio, con el objetivo de facilitar el entendimiento de los términos manejados en el diagrama.

CRed: controla el intercambio de mensajes entre las naves.

CNave: realiza los movimientos y las técnicas predictivas.

CPos: la posición que recibe cada nave.

3.2 -Modelo de casos de uso del sistema:



3.2.1-Actores del Sistema

Actor	**Justificación**
Simulación	Es quien va a interactuar con los diferentes objetos que definen la simulación
Timer	Periódicamente actualiza las posiciones de los objetos.

3.2.2-Casos de uso del Sistema

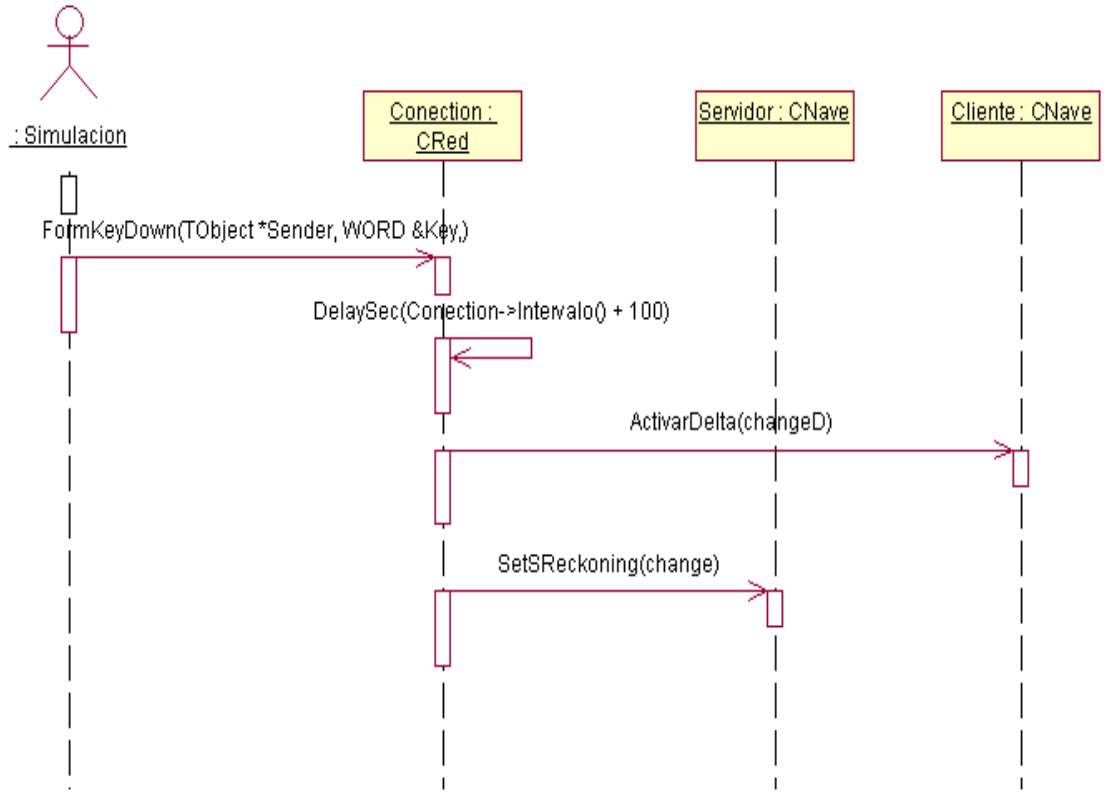
CU-1	Transmitir por la red
Actor	Simulación
Descripción	Este caso de uso permite transmitir los paquetes de información a través de la red, que contienen la información acerca de la posición y la orientación.

CU-2	Cambiar la configuración de las técnicas a demostrar
Actor	Simulación
Descripción	Este caso de uso permite cambiar la configuración, el orden o terminar la ejecución de alguna técnica de optimización empleada.

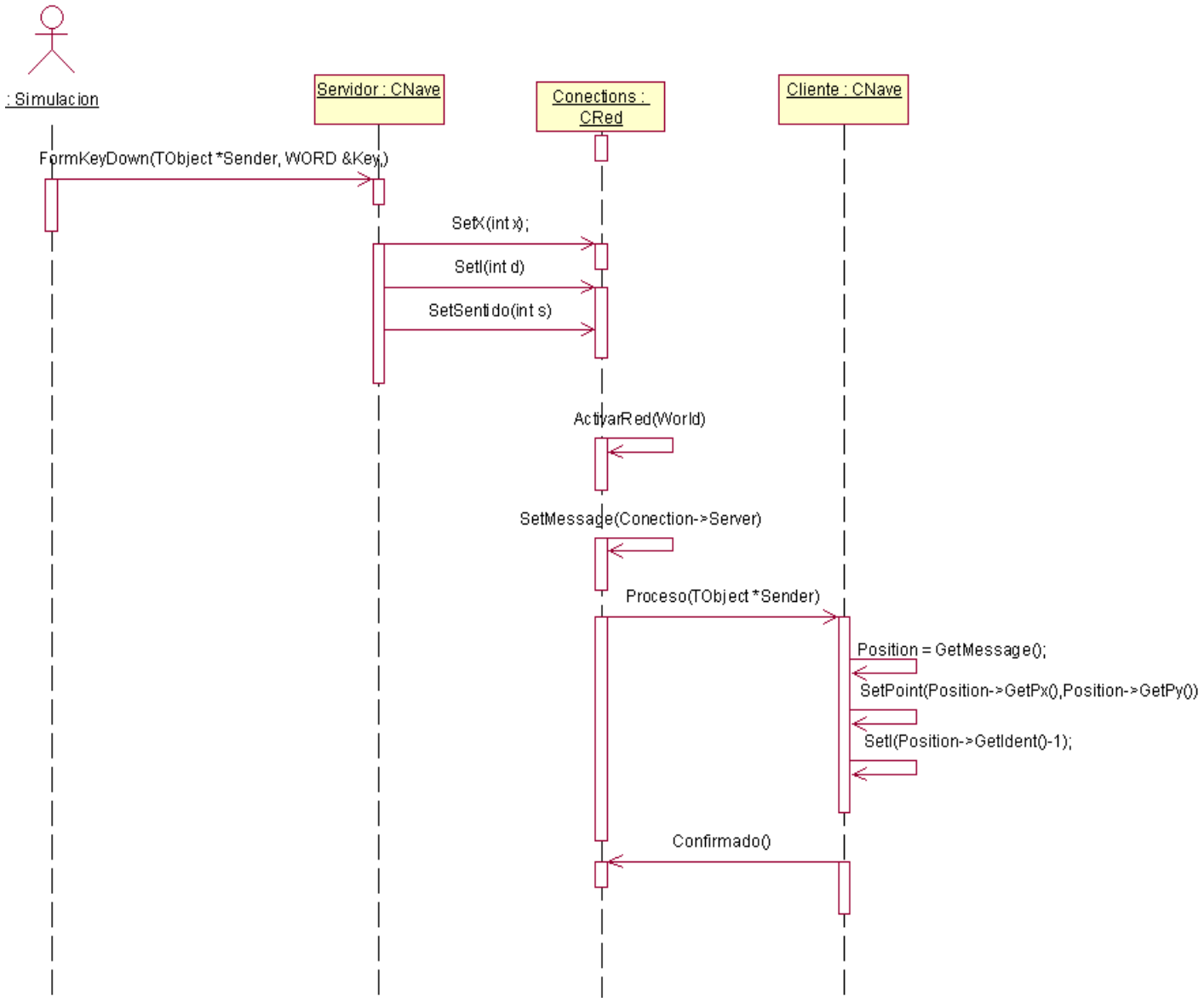
CU-3	Actualizar la información de la nave
Timer	Simulación
Descripción	Este caso de uso permite sincronizar la información que define la simulación (posición y orientación), periódicamente.

3.3 -Diagramas de secuencia

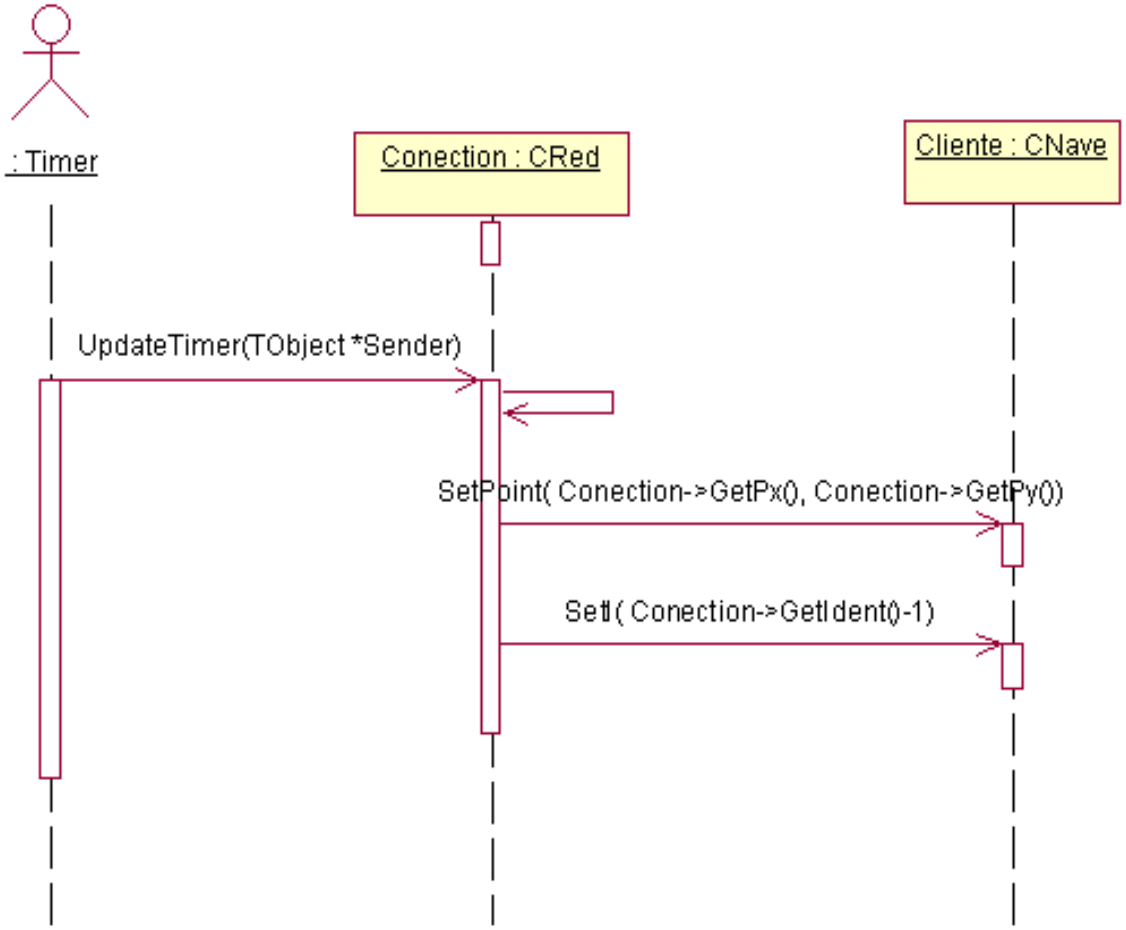
3.3.1 - Cambiar la configuración de la red



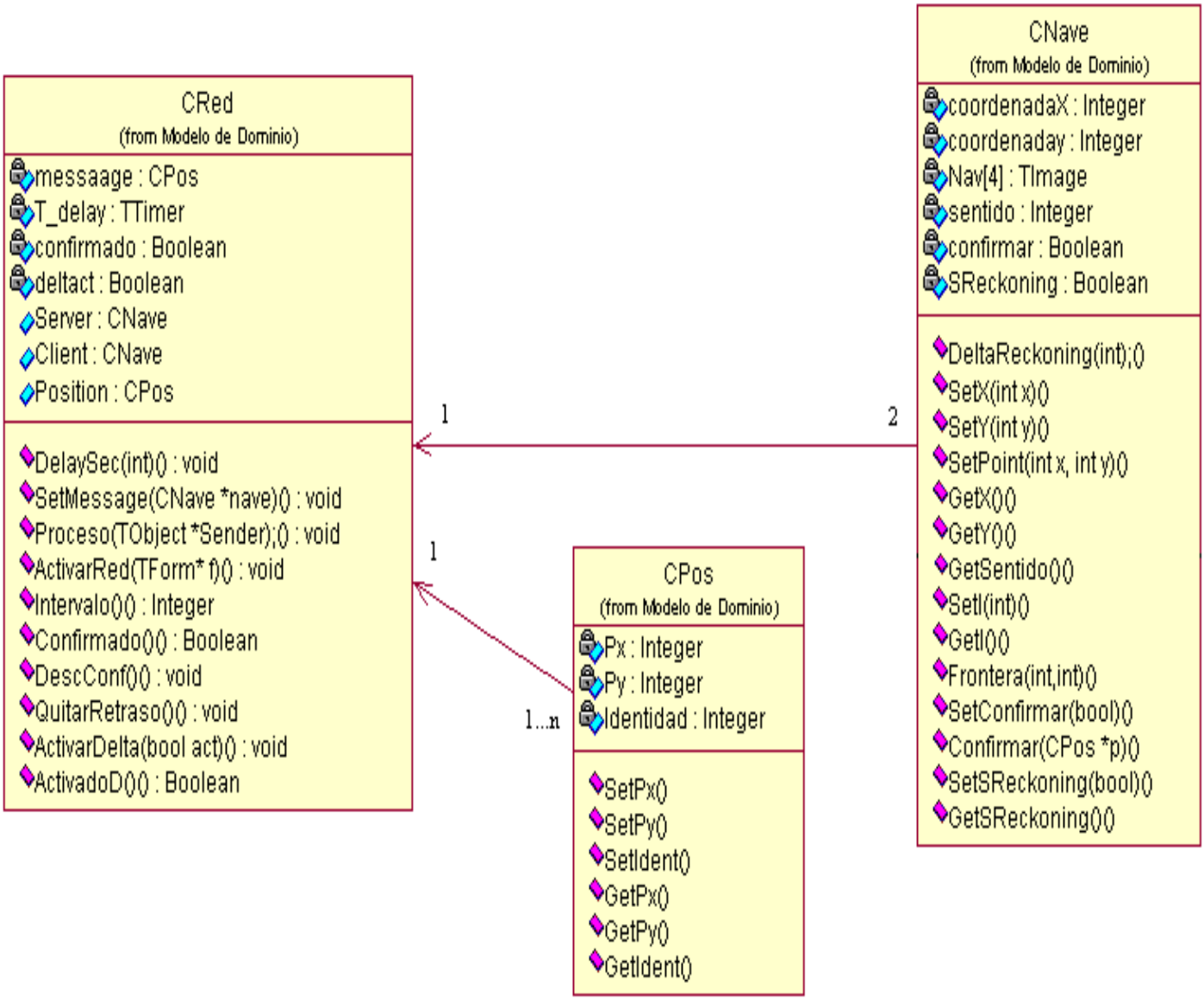
3.3.2- Transmitir por la red



3.3.3 – Actualizar la información



3.4- Diagrama de clases



3.5 -Descripción de las clases con sus atributos y principales funcionalidades.

3.5.1 –Clase CRed:

Nombre	CRed
Tipo de clase	Controladora

Atributo	**Tipo**
messageO	CPos*
T_delay	TTimer*
messages	TList*
confirmado	Boolean
deltact	Boolean
Server	CNave*
Client	CNave*
Position	CPos*

Para cada responsabilidad:	
Nombre:	DelaySec(int)
Descripción:	Devuelve el último mensaje recibido, desde la lista de mensajes.
Nombre:	Proceso(TObject *Sender)
Descripción:	Realiza el envió del mensaje con la posición y orientación hacia el cliente.
Nombre:	ActivarRed(TForm* f)
Descripción:	Activa el control de retraso.
Nombre:	Intervalo()
Descripción:	Devuelve el intervalo de tiempo del retraso.
Nombre:	Confirmado()
Descripción:	Devuelve el estado de la confirmación
Nombre:	DescConf()
Descripción:	Regresa el estado de la variable confirmado a su estado original, para que sea falso mientras no se vuelva a confirmar.

Nombre:	QuitarRetraso()
Descripción:	Elimina el retraso.
Nombre:	ActivarDelta(bool act)
Descripción:	Asigna a la variable deltact, el valor de act;. Para conocer si el delta ha sido activado en el cliente.
Nombre:	ActivadoD()
Descripción:	Devuelve el estado del atributo deltact.

3.5.2 –Clase CNave:

Nombre	CNave
Tipo de clase	Entidad

Atributo	**Tipo**
coordenadaX	Integer
coordenadaY	Integer
Nav[4]	TImage*
sentido	Integer
confirmar	Boolean
SReckoning	Boolean

Para cada responsabilidad:	
Nombre:	DeltaReckoning(int)
Descripción:	Recibe el sentido del movimiento actual de la nave, y asigna una nueva posición, basándose en la suposición de que mantendrá su velocidad y sentido actual.
Nombre:	Frontera(int,int)
Descripción:	Impide al objeto nave (Servidor), salirse de los límites del panel.
Nombre:	SetConfirmar(bool)
Descripción:	Asigna el estado de una variable booleana al atributo confirmar.
Nombre:	Confirmar(CPos *p)

Descripción:	Si la posición actual coincide con la posición que recibe, devuelve verdadero. De este modo se puede conocer si el cliente tiene la misma posición que el servidor, de lo contrario el servidor no podrá moverse, hasta que se le confirme.
Nombre:	SetSReckoning(bool)
Descripción:	Si se activa, el servidor asume que se le ha confirmado, y no tendrá que esperar confirmaciones del cliente.

3.5.3 -Clase CPos:

Nombre	CPos
Tipo de clase	Entidad
Atributo	**Tipo**
Px	Integer
Py	Integer
Identidad	Integer
Esta clase se encarga de encapsular las coordenadas x, y, más la orientación del objeto.	

Conclusiones

Las técnicas descritas son de aplicación opcional. Es aconsejable y posiblemente necesario adaptarlas a cada entorno, para un aprovechamiento más efectivo. Se debe ser cuidadoso a la hora de escoger cuales técnicas implementar y a la hora de corregir sus inconvenientes, que eventualmente se presentarán y de decidir si sus beneficios son superiores a sus inconvenientes. No debe verse una técnica como la solución a todos los problemas, sino la concatenación de varias, que derivarán en un alto impacto en la jugabilidad y el realismo del juego. Se ha hecho énfasis en las principales técnicas conocidas, muchas de ellas ya probadas e implementadas en juegos funcionales. Es posible la existencia de modificaciones de algunas o algoritmos novedosos que debido a su falta de documentación o dificultad de implementación hayan sido poco tratados, o no tratados del todo.

Recomendaciones

Dado el vertiginoso avance de la ciencia, es posible que este trabajo pierda actualidad de forma relativamente rápida, algunas técnicas podrían entrar en desuso, siendo sustituidas por otras más actuales. Podría ser necesario que se actualizara eventualmente el contenido de este documento para que pueda continuar siendo efectivo y útil. Podría a partir de este catálogo realizarse una especie de guía técnica o metodología que defina o recomiende cómo y que técnicas emplear en cada caso o problemática para cada tipo de juego, según sus especificidades. Podría ser también de interés de los proyectos de simuladores y juegos, de la facultad que implementan redes, mejorar dichas implementaciones con algunas de las técnicas propuestas en este trabajo.

Bibliografía:

1. **Neyland, David L.** *Virtual Combat: A Guide to Distribuited Interactive Simulations.* s.l. : Stackpole Books, 1997. ISBN: 0811731251.

2. **Bettner, Paul y Terrano, Mark.** GDC 2001: 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. *www.gamasutra.com.* [En línea] 22 de Marzo de 2001. http://www.gamasutra.com/features/20010322/terrano_01.htm.

3. *Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment.* **Park, Kyoung Shin y Kenyon, Robert V.** s.l. : Virtual Reality, 1999.

4. **Ng, Yu-Shen.** Designing Fast-Action Games . *Gamasutra.* [En línea] 5 de September de 1997.

5. **Epic MegaGames, Inc.** Unreal Networking Architecture. [Online] 07 21, 1999.

6. *Management of Networked Virtual Environments.* **Snyers d'Attenhoven, Jehan.** s.l. : University of Brussels, 2002.

7. **Greer, Jim and Boot, Zachary.** Minimizing Latency in real Time Strategic Games. *Game Programing Gems 3.*

8. **Aronson, Jesse.** Using Groupings for Networked Gaming. *Gamasutra.* [Online] June 21, 2000 . http://www.gamasutra.com/features/20000621/aronson_01.htm.

9. **Shi, Larry and Zhang, Sao.** Time and Consistence Management for Multiserver Based MMORPGs. *Game Programiing Gems 4.*

10. The Quake3 Networking Model. *Book Hook.* [Online]

11. **Haag, Chris.** Targeting:A variation of dead reckoning. *GameDev.net.* [Online] 5 17, 2001 .

12. **Ramskov, Jakob.** Statistical Client Prediction. *GameDev.net.* [Online] Febrero 12, 1999 .

13. **Olsen, Jhon.** Interpolation Methods. *Game Programing Gems 1.*

14. **Aronson, Jesse.** *Gamasutra.* [En línea] CMP Media LLC, 19 de Septiembre de 1997.

15. **J., Smed, T., Kaukoranta y H., Hakonen.** *Aspects of networking in multiplayer computer games .* s.l. : Emerald Group Publishing Limited, 2002.

16. *An end-to-end communication architecture for collaborative virtual environments.* **Shirmohammadi, Shervin y Georganas, Nicolas D.** 2-3, s.l. : Computer Networks, 2001, Vol. 35.

17. **Smed, Jouni and Harry, Hakonen.** *Algorithms and Networking for Computer Games.* West Sussex : John Wiley & Sons Ltd, 2006. ISBN-13: 978-0-047-01812-5.

18. **Ramirez, Ramiro A.** http://www.peiper.com.ar/edicion06/compresion.pdf. *www.peiper.com.ar.* [En línea] 31 de Marzo de 2008.

Anexos:

Anexo 1:

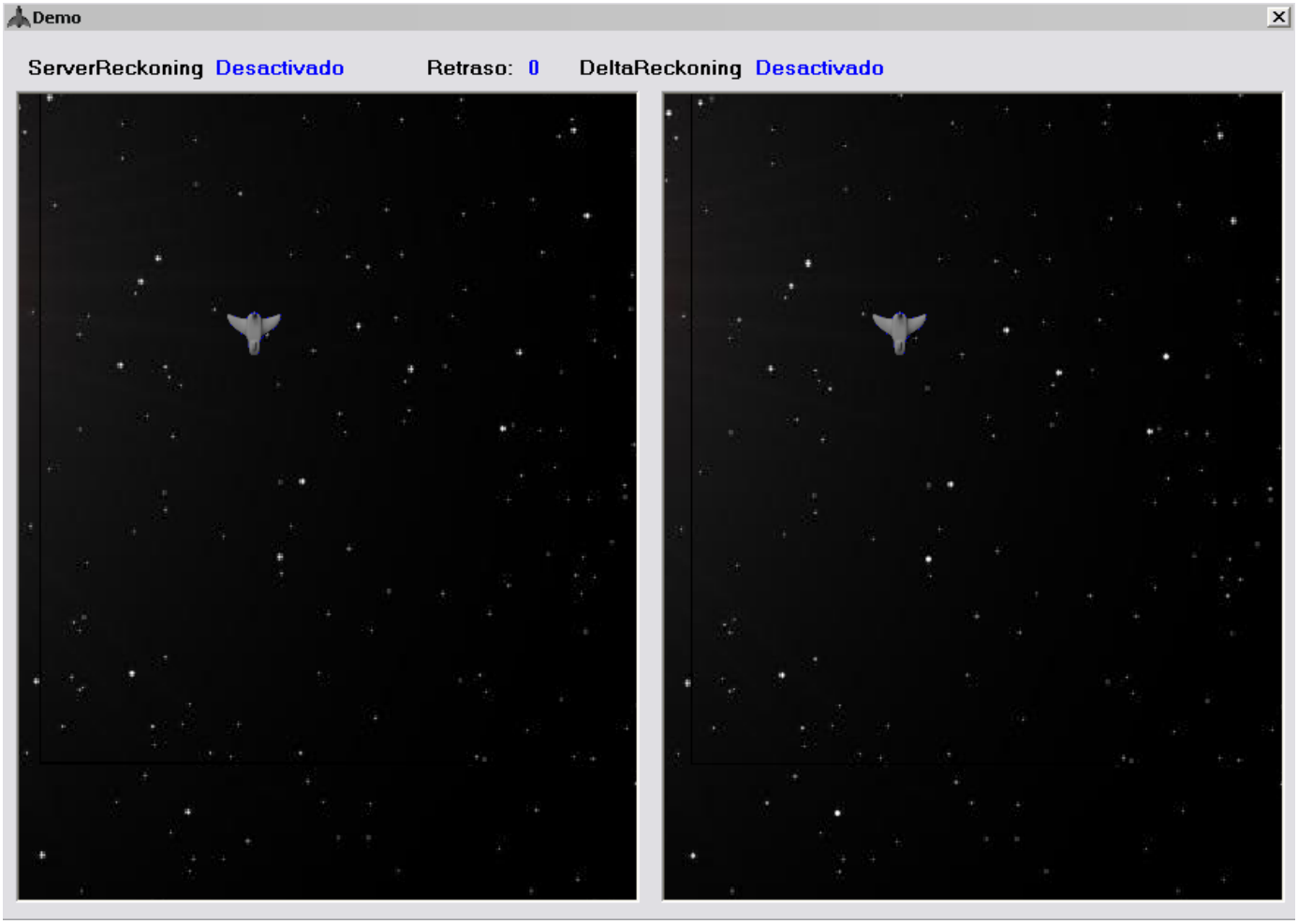


Figura 20: Condiciones ideales.

Anexo 2:

Figura 21: Retraso impuesto a los objetos.

Anexo 3:

Figura 22: ServerReckoning en acción.

Anexo 4:

Figura 23: ServerReckoning y DeltaReckoning evitando el retraso.

Anexo 5:

Figura 24: Inconveniente del DeltaReckoning

Glosario:

Juegos distribuidos: juego cuya simulación ocurre en varios nodos al mismo tiempo.

Tiempos de respuesta: tiempo que transcurre desde que una acción es ordenada o enviada hasta que es ejecutada.

Consistencia: los datos que definen la simulación de un objeto específico en más de un nodo son iguales o equivalentes.

Escalabilidad: propiedad de los servidores de admitir nuevos clientes.

Extensibilidad: es la capacidad de agregar o modificar el comportamiento de objetos.

Búsqueda de caminos: path finding (por sus siglas en ingles), conjunto de algoritmos diseñados para que entidades no controladas por humanos puedan descubrir y recorrer trayectorias en entornos virtuales.

Avatar: extrapolación del usuario humano al entorno virtual en forma de marioneta.

Entropía: Es también la cantidad de "ruido" o "desorden" que contiene o libera un sistema y es una medida de la información contenida en el mensaje. La entropía nos indica el límite teórico para la compresión de datos.

Algoritmo Huffman: se refiere al uso de una tabla de códigos de longitud variable para codificar un determinado símbolo. Algoritmo de compresión de datos muy eficiente.