**Sistemas Distribuidos**

**Profesor José Octavio Gutiérrez García**

**Proyecto Alpha**

Aline Becerra Carranza 149958

Maritrini García Ruiz 151490

**Planteamiento**

El Proyecto Alpha consiste en la implementación del juego “Pégale al monstruo”. Las especificaciones del juego y flujo del mismo radican en:

* Cada cliente en una partida tiene una interfaz con una rejilla en la cual se desplegarán los monstruos que le sean mandados.
* El servidor envía a los clientes, por medio de un mensaje multicast, un “monstruo” y la posición en la cual aparecerá.
* El monstruo es desplegado. El servidor necesita abrir una ventana de tiempo para recibir las respuestas de cada cliente.
* Las respuestas recibidas por el servidor le indican si el usuario ha golpeado al “monstruo”, este mensaje es enviado vía sockets TCP.
* El cliente que golpee cierto número de monstruos primero es el ganador.
* El servidor es el encargado de avisarles a los jugadores vía multicast quien ganó, para después inicializarse para empezar otro juego.
* Los clientes deben ser capaces de entrar y salir de una partida, sin afectar la partida.
* Para comenzar a jugar, es necesario que el cliente se registre, ya que, sólo los jugadores registrados podrán jugar. Este servicio de registro es implementado por el servidor y es desplegado en Java RMI.
* Una vez que el usuario se registró, el servidor le pasa una dirección IP y los puertos con los que se realiza el juego ( usando Java RMI) .

Éstas fueron las principales funcionalidades cubiertas y flujo abstraído dentro del programa desarrollado, en sus dos versiones. En el siguiente apartado se detalla la solución implementada como preámbulo para el hablar de las pruebas de desempeño y evaluación del proyecto.

**Solución implementada**

Se realizaron dos proyectos, “Proyecto Alpha Ejecutable” y “ Proyecto Alpha Estresamiento Tiempo”, ambos proyectos tienen la misma lógica subyacente. La diferencia entre ellos radica en que el primero permite ver el correcto funcionamiento de la solución desde la perspectiva del usuario final y el segundo fue realizado con propósitos de evaluación y análisis del desempeño del sistema cuando se tenga una cantidad distinta de clientes al mismo tiempo. Para este último, se analizó el tiempo de la respuesta promedio y su desviación estándar.

La implementación del proyecto está dividida en cuatro paquetes: paquete de cliente, que cuenta con las clases ClienteGUI y ClienteMulticast; el paquete de comunicación,que cuenta con las clases Juego y Monstruo; el paquete de interfaces, que cuenta con la interfaz PegaleAlMonstruo y, por último, paquete de servidor que contiene las clases Partida,Servidor,ServidorMulticast y ServidorTCP.

Para empezar el juego es necesario correr la clase Servidor, que implementa la interfaz PegaleAlMonstruo, esta clase tiene como atributos datosJuego, puertoRMI, nombreServicio y Partida de tipo Partida (de esta clase hablará más adelante). En el servidor se definen los puertos que se usarán, tanto para multicast como para TCP; de igual manera, conoce la IP del servidor y la dirección multicast.

Una vez definidos los datos principales del juego, el servidor crea el registro con base en el puerto RMI dado y registra el servicio, en esta clase se instancian e inician los servidores multicast y TCP que se van a utilizar, éstos son dos hilos que se necesitan correr al mismo tiempo. Éstos reciben como parámetro el registro del stub, generado por el servidor para el servicio RMI PegaleAlMonstruo y la partida actual.

La clase ServidorMulticast, que extiende la clase Hilo y pertenece al paquete del servidor, recibe como parámetros una partida de tipo Partida y datos de tipo Juego; de la partida obtiene el número de celdas que se tienen, el puerto y la dirección para comunicación multicast. Ésta clase cuenta con los atributos partida, celdas, segundos, puerto,socketMulticast y una dirección. La clase cuenta con un método run(), en el cual, si aún no ha habido un ganador en la partida genera aleatoriamente un entero entre cero y celdas-1 para que representen el número de rejilla donde caerá el monstruo. Partiendo de este crea el mensaje, lo pasa al formato bytes y lo manda en un DatagramPackage .Otro mensaje que manda esta clase es cuando hay un ganador en la partida, en este caso avisa a todos los usuarios. El formato de los mensajes es “Ganador <usuario> fin” o bien “Monstruo <n°de celda> fin”. Se le agrega un fin al final, pues al convertirlo a String del lado del cliente, puede agregar dígitos como 1 ó 0, que generaba problemas, sobre todo, en la versión de estresamiento.

La clase Servidor TCP también recibe como parámetros una partida de tipo Partida y datos de tipo Juego. La clase tiene como atributos una partida, un int que representa un puerto y un socketEntrada de tipo ServerSocket, el puerto TCP a utilizar es conocido gracias a los datos de tipo Juego. Además, cuenta con un método run() que instancia un socketCliente de tipo Socket , una conexión de tipo Conexión que recibe un socketCliente y una partida y, finalmente, empieza la conexión. La clase Connection implementada en la clase ServidorTCP recive un socket y una partida y con base en esto crea los puentes para DataUnputStream y el DataOutputStream. En el método run() de esta misma clase lee los mensajes que llegan del usuario, aquí se valida si el usuario “golpeo” o no de forma correcta al monstruo y en caso de que sí aumenta la puntuación del usuario y posteriormente cierra el socket. Lo que se hace, en resumen, es que siempre se tiene el hilo principal abierto que obtendrá las conexiones y a partir de ésta se crean hilos hijos cada que llega un mensaje por parte del cliente, éstos realizan la acción de validación antes mencionada, y mueren. Respecto a la forma en la que se agregan los puntos se emplean funciones synchronized pues se tienen variables compartidas que están siendo modificadas por distintos hilos, esto se realizó en la clase Partida.

La clase Partida también se encuentra en el paquete Servidor y contiene todos los datos relevantes para la partida, sus atributos son un hashmap de puntuaciones, los puntos necesarios para ganar la partida, un int celdaMonstruo que muestra donde se encuentra el mosntruo actualmente y un String que contiene al ganador.Una vez que el usuario se registra, la clase Partida es la encargada de asignarle la puntuación dependiendo si es un usuario nuevo o si es un usuario que reingresa a la partida; de igual manera, se encarga de modificar la celda en la que se encuentra el monstruo. Esta clase conoce las puntuaciones de todos los usuarios, es capaz de agregarle puntos a un usuario o reiniciar su puntuación, igualmente es capaz de saber quien es el ganador de la partida actual, éste es uno de los métodos sincronizados.

Por último, el servidor invoca el método registroJuego() que recibe como parámetro un usuario, despliega un mensaje dependiendo si es un usuario de nuevo ingreso o si volvió a la partida y ,en seguida, instancia la clase Juego perteneciente al paquete de comunicación. A éste se le pasa como parámetro la dirección IP del servidor, el puerto que será utilizado para TCP, el puerto que será utilizado para multicast, la dirección multicast y el número de celdas .

En el paquete de comunicación, se encuentran dos clases Monstruo y Juego. La clase Monstruo tiene únicamente el atributo celda . La clase juego tiene como atributos la dirección IP, los puertos para TCP y Multicast, la dirección multicast y el número de celdas e implementa getters y setters para todos estos atributos. Este paquete está pensado para tener las clases que utilizan tanto servidor como usuario para su correcto funcionamiento y comunicación entre ellos.

El paquete cliente varía dependiendo de si se habla de la versión estresamiento o de la versión ejecutable. Aquí se hablará de la versión ejecutable. En este paquee se encuentra el diseño del GUI, que permite al usuario registrarse y después empezar al juego.

La clase ClienteGUI cuenta con las variables usuario de tipo String ,int puntuación, Juego juegoInfo, PegaleAlMonstruo servicio Juego, HashMap de tipo Integer para las celdas, s de tipo socket e in y out de tipo DataInputStream. Esta clase permite el registro o reingreso de un usuario, esto por medio del método remoto en RMI. Una vez que se da el paso anterior exitosamente, se despliega la interfaz para el juego. La clase ClienteMulticast implementa la clase hilo y tiene como atributos un hashmap de celdas, una variable tipo Juego con información del Juego, un entero que representa la celda activa, un botón de salir, un usuario , las puntuaciones, un socketMulticast y un boolean ganador. Esta clase permite que mientras no exista un ganador se une al grupo multicast y recibir los mensajes que aquí sean enviados, estos mostrarán el que sean monstruos (y por lo tanto cambio la interfaz) o bien hay ganador y despliega el mensaje conveniente. Regresando a la clase ClienteGUI cuando un usuario le pega a un monstruo, se genera y envía un mensaje al servidor vía TCP gracias el método enviaMensaje(); este método crea un socket con la dirección IP y el puerto TCP obtenidos de la clase Juego , manda la respuesta del usuario y recibe la respuesta del servidor.

Finalmente, en la siguiente figura se muestra el esquema del Proyecto Alpha Ejecutable en Netbeans.

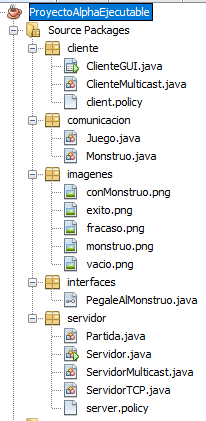


Figura1. Esquema del proyecto versión ejecutable.

**Evaluación de desempeño: Pruebas realizadas**

Para la evaluación del desempeño del programa desarrollado, se desarrolló un experimento basado en pruebas. En este apartado se desarrollan las mismas. los resultados obtenidos en éstas y su análisis. Las primeras pruebas incluyeron el análisis sobre lógica de la implementación y funcionamiento por componentes y por interacción de los mismos como un todo. Posteriormente, se vio el desempeño del juego implementado, desde el aspecto de un entorno de múltiples clientes y desde el del uso solo con la interfaz definida y desde un equipo distinto a aquel donde se ha desplegado el servidor del juego. Cada una de éstas se desarrolla en siguientes apartados.

***Pruebas unitarias***

Éstas hacen referencia a la primeras etapas de implementación de la solución. En éstas pruebas, se probó el correcto funcionamiento de cada uno de los componentes principales del programa desarrollado, éstos son el servidor, cliente e interfaz de juego.

En el caso del cliente y el servidor, los módulos que se probaron fueron, principalmente, los relacionados con los protocolos de comunicación empleados (Multicast y TCP) y mecanismo de invocación remota (Java RMI). Para lo anterior, se utilizaron los proyectos desarrollados en clase. Un ejemplo de lo anterior fue el usar la clase *MulticastReceivingPeer* desarrollada en clase para corroborar el correcto envío del mensaje Multicast por parte del cliente.

De éstas se pudieron detectar y corregir errores en la lógica implementada, lo cual culminó en la obtención del correcto funcionamiento de las partes. El siguiente paso natural fue el de corroborar su funcionamiento como un todo.

***Pruebas de integración***

En esta etapa de pruebas, se desarrolló un proyecto base. Éste integraba la funcionalidad de servidor y cliente sin una interfaz aún. El objetivo de éste era ser un punto base a partir del cual se desarrollara su versión estresamiento y versión ejecutable.

De esta etapa, surgieron diversos errores que pudieron ser solucionados. Éstos fueron los siguientes.

* *Multicast en sistema operativo Mac OS X.* Debido a preferencias distintas dependiendo del sistema operativo, en éste se presenta el problema del correcto despliegue de multicast. Para su solución se agregó la línea System.setProperty("java.net.preferIPv4Stack", "true") en el main principal del cliente y servidor.
* *RMI de manera remoto.* Éste se presentó al querer realizar la invocación a un método remoto, en una computadora distinta a la local. Esto se soluciona a través de la línea System.setProperty("java.rmi.server.hostname", <String de la IP del host de los servicios de RMI>) la cual tuvo que colocarse en el main del servidor donde se despliega RMI.

***Funcionamiento automático***

Una vez que se tenía la solución base, mencionada con anterioridad, se procedió a realizar la versión de estresamiento del proyecto. Para esto se utilizó la misma implementación del proyecto, sólo se agregaron las clases Cliente, ClientThread, ClientRespuesta, y la clase Launcher, en lugar de las clases ClienteGUI y ClienteMulticast. Las clases Cliente y ClienteRespuesta son las principales; el Launcher sólo se encarga de lanzar los ClientThreands que en su método run() instancia Clientes.

La clase Cliente implementa la funcionalidad del cliente respecto RMI después de realizado el registry, este útlimo sólo se hace una vez al inicio y se pasa como parámetros a todos los clientes, y la clase ClienteRespuesta implementa toda la funcionalidad TCP del cliente.

La clase Launcher tiene como atributos el número de clientes a participar, la dirección RMI y el registry, al momento de realizar las pruebas de estresamiento el atributo número de clientes fue cambiado por los valores 25, 50,100, 150, 200, 300 y 400, este valor determina el número de hilos que serán creados y llamados con el método start() de ClientThread.

En la clase ServidorMulticast se establece el tiempo que pasa entre monstruo y monstruo, en la versión ejecutable este valor se estableció en 1.5 segundos, sin embargo, en la versión de estresamiento este valor se estableció en 3 segundos para que hubiera holgura en el registro de los clientes, pues si este valor no era aumentado pasaba que ya había un ganador cuando ciertos usuarios todavía no se habían registrado.Esto de igual manera se da porque el funcionamiento de este modo de estresamiento se basa en crear un número de pseudo usuarios inteligentes que contestan “bien” en todas las ocasiones y de manera inmediata después de recibir el mensaje multicast y van aumentando su puntaje de uno en uno hasta llegar a 5 puntos. El primer usuario que llegue a este puntaje, según el servidor, es seleccionado como el ganador.

En la siguiente figura se muestra el esquema del proyecto en Netbeans.

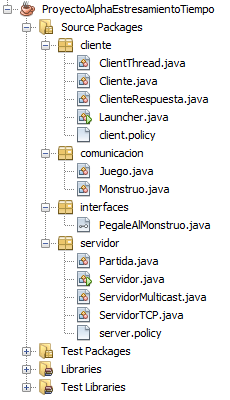


Figura2. Esquema del proyecto versión estresamiento

El objetivo de esta prueba de estresamiento era conocer las estadísticas relacionada con los tiempos promedios presentes entre el envío de un mensaje multicast y la recepción de su respuesta en TCP. En el juego Pégale al Monstruo esto se ve en el tiempo desde que el servidor envía un mensaje multicast con la posición del monstruo y la recepción del mensaje TCP respuesta del cliente anterior. Los datos obtenidos en el análisis de estos tiempos están representados a continuación en tablas y gráficas. La Figura3 muestra la tabla del tiempo promedio de respuesta (ms) dependiendo el número de clientes involucrados en el juego. La Figura4 muestra la gráfica de dicha tabla. La Figura5 muestra la tabla de la desviación estándar promedio de la respuesta en (ms) dependiendo el número de clientes involucrados en el juego. Finalmente, la Figura6 muestra la gráfica de dicha tabla.

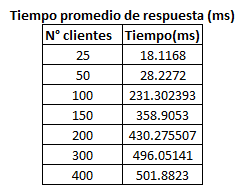


Figura3. Tiempo promedio de respuesta (ms)

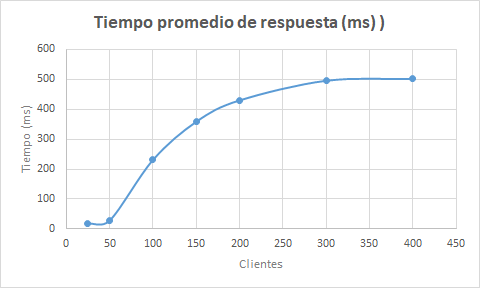


Figura4. Tiempo promedio de respuesta (ms)

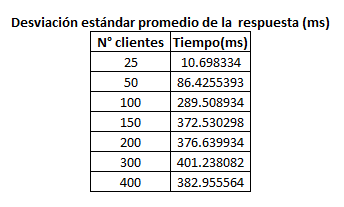


Figura5. Desviación estándar promedio de respuesta (ms)

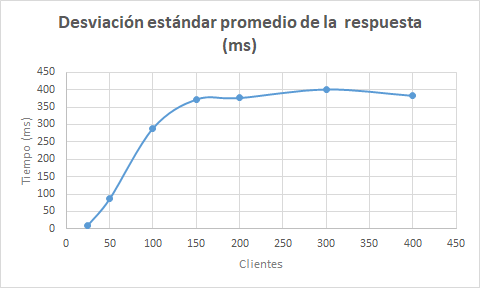


Figura6. Desviación estándar promedio de respuesta (ms)

En la Figura4 se observa el aumento que presenta el tiempo promedio en milisegundos con respecto al aumento de clientes en el sistema, este aumento tiene un comportamiento exponencial. Sin embargo, a partir de los 200 clientes este crecimiento exponencial empieza a presentar un comportamiento asintótico, esto se debe a que a partir de los 200 clientes se empezaron a generar errores en el programa sobre error de conexión en los sockets, se está llegando al límite de la capacidad para la comunicación esto no cambiará para mayor cantidad de clientes y sólo hará que a partir de los 200 los errores sigan aumento, inclusive esto se dará con comportamiento exponencial.

En la Figura6. se muestra que la desviación estándar promedio de la respuesta en milisegundos, es importante notar que las gráfica de la desviación estándar es muy parecida a la del tiempo promedio de respuesta. Esto ya que los tiempos que se presentan van desde uno hasta un número mucho mayor a la media. La desviación no está significativamente acotada en cada caso y depende de la situación en la que se observe, absorbe el comportamiento de ese experimento.

***Funcionamiento manual***

Finalmente, en el aspecto de desempeño a nivel usuario del sistema, se realizaron, principalmente dos pruebas. La primera consistió en la corrida de una partida con uno o múltiples clientes que acceden y salen del juego de manera dinámica en la misma PC donde se ha levantado el servidor. La segunda hacía referencia al mismo mecanismo de juego pero los usuarios se encontraban en equipos distintos a donde se había levantado el servidor.

En la primera prueba, lo que se hizo fue levantar el servidor y correr de uno a cinco clientes para la misma partida. Se comprobó el correcto funcionamiento del programa, esto ya que:

* El despliegue del servicio remoto y el acceso al mismo se hizo de manera fluída y sin errores de conexión o retardo en la respuesta.
* La ventana de tiempo acordada por el servidor (1.5 s) para la visibilidad de un monstruo empata con lo mostrado en la interfaz.
* Se mostró el correcto registro y reingreso a una partida por parte de los usuarios. Si alguna de estas acciones se realizaba en uno de ellos no interfería en las actividades de los demás.
* Se corroboró el correcto despliegue del mensaje de fin de partido dependiendo quien había sido el ganador.

En conclusión, se obtuvo una primera prueba exitosa y satisfactoria. Esto principalmente a que ya se habían solucionado ciertos problemas en puntos anteriores y que todo se estaba haciendo en localhost.

En la segunda prueba, se buscaba realizar una prueba más realista del funcionamiento del programa, pues en general se espera que los clientes se encuentren de manera remota. El único cambio realizado respecto al planteamiento anterior En ésta se presentó el problema de que no se pudiera realizar la conexión por bloqueo en los puertos de alguno de los dos equipos. En estos casos se tuvo que recurrir al cambio del equipo y seguir en búsqueda de una solución para el futuro. Omitiendo el pasado aspecto se obtuvo el mismo funcionamiento que en la primera prueba.

***Conclusiones***

Como conclusiones generales, creemos que realizar este proyecto fue muy gratificante, pues, en él, pusimos en práctica muchos conocimientos adquiridos en clase y nos dimos cuenta lo mucho que hemos aprendido. Igualmente, el proyecto representó un reto y nos impulsó a ir más allá de lo visto en clase, investigando comandos necesarios para arreglar ciertos errores, por ejemplo, el número máximo de datagram sockets posibles en una computadora o ciertas líneas de código necesarias cuando se cambia de sistema operativo. Comprobamos el correcto funcionamiento del proyecto ejecutable, pues implementó todos los puntos de los requerimientos y su funcionamiento fue el esperado , un punto de posibles mejoras sería que el usuario pueda introducir un nombre separado por espacios, pues si se introduce uno en la implementación actual el proyecto manda errores .

En cuanto a la versión de estresamiento, los resultados obtenidos fueron los esperados, pues, el promedio del tiempo de respuesta aumenta de manera exponencial conforme se aumenta el número de usuarios en la partida y esto empata en el comportamiento de la varianza, realizar este estresamiento del proyecto nos permitió conocer las limitaciones del mismo pues así podemos asegurar el correcto funcionamiento para menos de 200 usuarios y a partir de ahí saber que los errores incrementarán más y más y por esto mismo se creará una asíntota en el tiempo entre el envío de un mensaje y la recepción del mismo.