



4-10-2024

Aplicaciones para comunicaciones  
en red

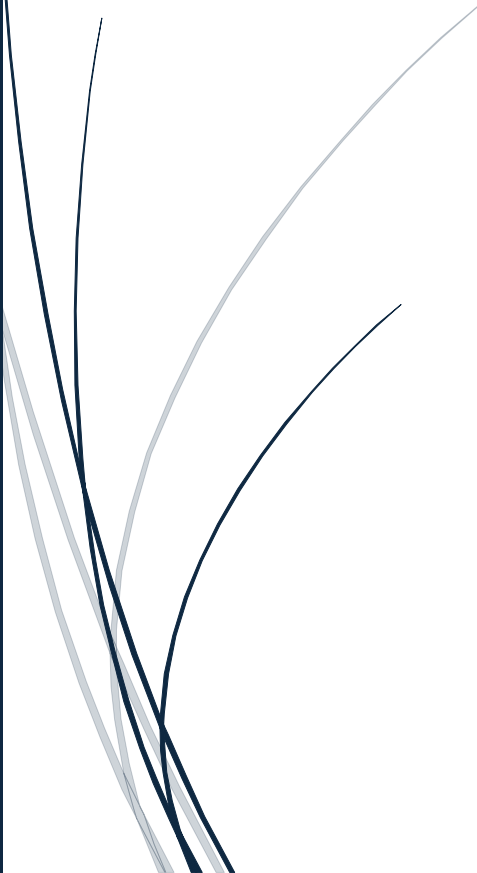
Practica 1

**Integrantes:**

Flores Morales Aldahir Andrés

Velasco Jimenez Luis Antonio

**Grupo:** 6CM1



## Resumen

Durante esta práctica en la que se tenía que implementar el juego Buscaminas, hemos entendido la importancia de la comunicación de los sockets de flujo bloqueantes, ya que hemos tenido que trabajar con la arquitectura Cliente-Servidor, donde nuestro cliente puede jugar y hacer una acción en el tablero, mientras el servidor procesará el movimiento. Pusimos en práctica teoría de comunicaciones en red entre cliente y servidor, con el objetivo de poder entablar esto en código, y crear el juego Buscaminas tradicional de 3 niveles distintos.

## Introducción

La comunicación por sockets permite que dos dispositivos se comuniquen a través de una red utilizando un canal bidireccional. Por defecto, los sockets operan en un modo de bloqueo (bloqueantes), lo que significa que cuando una operación de entrada/salida se ejecuta, el proceso se detiene hasta que la operación se completa. Este enfoque, aunque sencillo, no es óptimo para aplicaciones que necesitan interactuar con múltiples clientes a la vez o que requieren que el flujo de datos sea continuo y ágil, como en un juego en red.

En contraste, los sockets de flujo no bloqueantes permiten que el programa continúe ejecutándose incluso cuando las operaciones de entrada/salida están en progreso. Esto se logra mediante el cambio del socket a un modo no bloqueante, de modo que si una operación no se puede completar de inmediato (por ejemplo, si no hay datos disponibles para leer), el programa sigue ejecutándose sin quedar detenido.

Cuando un socket no está en modo bloqueante, las operaciones de entrada/salida (como read o write) pueden devolver inmediatamente sin completar su acción, en lugar de detener el flujo hasta que se complete. Esto permite que el servidor gestione múltiples clientes en paralelo sin bloquear el programa en ninguna operación.

## Marco teórico

### Protocolos de comunicación

Los sistemas cliente-servidor dependen en gran medida de los protocolos de comunicación para facilitar la transferencia de datos entre dispositivos. Estos protocolos establecen las reglas y los procedimientos para la transmisión de información, asegurando que la comunicación sea eficiente, confiable y segura. Algunos de los protocolos más importantes aplicados en estos sistemas:

**TCP/IP (Transmission Control Protocol/Internet Protocol):** Este protocolo es fundamental en Internet y en los sistemas cliente-servidor. TCP garantiza una conexión confiable al dividir los datos en paquetes y asegurarse de que se entreguen correctamente, mientras que IP se encarga de direccionar los paquetes a través de la red.

**HTTP (Hypertext Transfer Protocol):** Ampliamente utilizado en la web, HTTP define cómo se comunican los navegadores web y los servidores. Establece un formato para las solicitudes y respuestas, permitiendo la transferencia de recursos como páginas web, imágenes y otros archivos.

**FTP (File Transfer Protocol):** Especializado en la transferencia de archivos, FTP facilita la carga y descarga de archivos entre un cliente y un servidor. Proporciona un método seguro y controlado para gestionar archivos remotos.

**SMTP (Simple Mail Transfer Protocol):** Esencial para el envío de correos electrónicos, SMTP define cómo se transmiten los mensajes de correo entre servidores. Asegura que los correos se entreguen de manera confiable y eficiente.

**SSH (Secure Shell):** Utilizado para acceder de forma segura a servidores remotos, SSH cifra la comunicación entre el cliente y el servidor, protegiendo los datos sensibles durante la transmisión.

Estos protocolos son fundamentales para el funcionamiento efectivo de los sistemas cliente-servidor, ya que permiten la comunicación fluida y segura entre dispositivos. Desde la transferencia de archivos hasta el intercambio de correos electrónicos, estos protocolos juegan un papel crucial en numerosos aspectos de la interacción en línea. Su implementación adecuada garantiza una experiencia de usuario óptima y protege la integridad de los datos en todo momento. En general los protocolos de comunicación son el corazón de los sistemas cliente-servidor, proporcionando el marco necesario para la transmisión confiable de información en esta era, sin embargo, para el desarrollo de esta práctica únicamente nos enfocamos en el protocolo FTP y TCP.

### Socket

Un socket, en el contexto de las redes informáticas, es un concepto fundamental que sirve como interfaz de comunicación entre dos programas que se ejecutan en la red. Esencialmente, actúa

como un punto final de una conexión entre dos nodos de una red, permitiendo la transferencia de datos de manera bidireccional

Este término proviene del mundo físico, donde un socket es el lugar donde se conectan los dispositivos eléctricos para recibir energía o comunicarse. En el ámbito de las redes informáticas, un socket es una abstracción que representa un extremo de una conexión de red, permitiendo que los programas se comuniquen entre sí, ya sea en la misma máquina (comunicación interna) o en diferentes máquinas (comunicación externa a través de la red).

Los sockets son una parte integral de la arquitectura de red de los sistemas informáticos modernos y son utilizados por una amplia gama de aplicaciones, desde navegadores web hasta aplicaciones de mensajería instantánea y servidores de archivos. Facilitan la comunicación entre procesos de manera eficiente y flexible, lo que permite la creación de aplicaciones y sistemas distribuidos complejos.

En términos de programación, los sockets se implementan mediante API (Interfaz de Programación de Aplicaciones) proporcionadas por el sistema operativo. Estas API permiten a los desarrolladores crear y gestionar sockets, establecer conexiones, enviar y recibir datos, y cerrar conexiones cuando ya no son necesarias.

Los sockets pueden ser de diferentes tipos, dependiendo del protocolo de transporte que utilicen. Los más comunes son los sockets de flujo (stream sockets) y los sockets de datagrama (datagram sockets). Los sockets de flujo utilizan el protocolo TCP (Protocolo de Control de Transmisión) para establecer una conexión orientada a la conexión y garantizar la entrega ordenada de datos, mientras que los sockets de datagrama utilizan el protocolo UDP (Protocolo de Datagrama de Usuario) y ofrecen una comunicación sin conexión y no garantizan la entrega ordenada de datos. Aunque en este caso para esta practica nos enfocamos principalmente en los sockets de flujo.

En general, un socket es una interfaz de comunicación que permite a los programas intercambiar datos a través de una red, ya sea en la misma máquina o en máquinas remotas. Es una abstracción fundamental en el desarrollo de aplicaciones de red y proporciona la base para la implementación de diversos servicios y protocolos de comunicación. Sin los sockets, la comunicación entre programas en una red sería extremadamente difícil, si no imposible, lo que subraya su importancia en el ámbito de la informática moderna.

### Sockets de flujo

Los sockets de flujo, una implementación clave en el ámbito de las redes informáticas, se destacan por su capacidad para facilitar la comunicación confiable y ordenada entre programas a través de una red. Estos sockets, basados en el protocolo TCP (Protocolo de Control de Transmisión), permiten establecer conexiones orientadas a la conexión entre dos puntos finales en la red. Una de las características principales de los sockets de flujo es su enfoque en la conexión orientada a la conexión. Antes de iniciar la transferencia de datos, se establece una conexión entre el cliente y el servidor mediante un proceso de establecimiento de conexión. Durante este proceso, se intercambian mensajes de control para sincronizar la comunicación y establecer un canal bidireccional para la transmisión de datos.

La confiabilidad y la integridad de los datos son aspectos críticos en las comunicaciones a través de sockets de flujo. El protocolo TCP garantiza una transferencia de datos confiable y ordenada mediante la implementación de mecanismos de control de flujo, retransmisión y verificación de entrega. Esto asegura que los datos lleguen al destino en el mismo orden en que fueron enviados y sin pérdida de información, incluso en condiciones de red adversas.

Otra característica importante de los sockets de flujo es su capacidad para proporcionar un flujo de datos continuo entre el cliente y el servidor. Esto significa que los datos se transfieren de manera fluida y sin interrupciones a través del canal establecido, lo que permite una comunicación eficiente y sin problemas entre los programas.

Los sockets de flujo también incorporan algoritmos de control de congestión para gestionar el flujo de datos y evitar la saturación de la red. Estos algoritmos ajustan dinámicamente la tasa de transmisión de datos en función de las condiciones de la red, evitando la congestión y optimizando el rendimiento de la comunicación.

Además, los sockets de flujo admiten la transmisión bidireccional de datos, lo que permite que tanto el cliente como el servidor envíen y reciban datos simultáneamente. Esta capacidad es fundamental para aplicaciones interactivas y en tiempo real, como videoconferencias, juegos en línea y sistemas de chat, donde la comunicación bidireccional es esencial.

En general, los sockets de flujo son una herramienta fundamental en el desarrollo de aplicaciones de red, proporcionando una conexión confiable, ordenada y continua entre programas a través de una red. Su capacidad para establecer conexiones orientadas a la conexión, garantizar la integridad de los datos y gestionar el flujo de datos los convierte en una opción ideal para una amplia gama de aplicaciones, desde la transferencia de archivos hasta la comunicación en tiempo real.

### Sockets bloqueantes

Los sockets bloqueantes son un componente fundamental en el desarrollo de aplicaciones de red. Estos sockets, también conocidos como sockets síncronos, operan de manera que una operación de lectura o escritura en el socket bloquea el hilo de ejecución hasta que la operación se completa. Este enfoque garantiza que las operaciones de entrada y salida se realicen de forma secuencial y sincrónica, lo que simplifica el manejo de la concurrencia en la aplicación.

En el contexto de la programación de redes, los sockets bloqueantes ofrecen una forma sencilla y directa de comunicación entre aplicaciones cliente y servidor. Cuando un cliente realiza una solicitud de conexión a un servidor a través de un socket bloqueante, el hilo de ejecución del cliente se bloquea hasta que se establece la conexión o se produce un error. Una vez establecida la conexión, las operaciones de lectura y escritura en el socket bloquearán el hilo de ejecución hasta que se completen con éxito.

Una de las principales características de los sockets bloqueantes es su capacidad para manejar una sola conexión a la vez. Esto significa que, en un entorno de servidor concurrente, cada solicitud de conexión bloquea el hilo de ejecución del servidor hasta que se atiende completamente. Si se reciben múltiples solicitudes de conexión simultáneas, el servidor debe administrar cada una de ellas de forma secuencial, lo que puede provocar un rendimiento deficiente en entornos de alto tráfico.

A pesar de estas limitaciones, los sockets bloqueantes siguen siendo muy utilizados en el desarrollo de aplicaciones de red por su simple aplicación. Para mitigar los problemas de rendimiento asociados con la concurrencia, es común emplear técnicas como el uso de múltiples subprocesos o la multiplexación de entrada/salida (E/S) para gestionar varias conexiones de manera eficiente.

## Desarrollo y ejecución

En general, el programa actúa pidiéndole el username y la dificultad con la que el usuario quiere jugar, esta información es adquirida desde el cliente, posteriormente el cliente establece una dirección y un puerto, en seguida le manda los datos de la dificultad y el servidor se encarga de crear un tablero dependiente a esta:

- Fácil: tablero de 9x9 con 10 minas
- Intermedio: tablero de 16X16 con 40 minas
- Experto: tablero de 16x30 con 99 minas

El usuario tiene la posibilidad de marcar en el tablero las casillas donde crea que haya minas, el numero de banderas es igual al numero de minas.

Si el usuario pierde, se le notifica con una ventana emergente y todas las minas serán mostradas por el tablero.

Si el usuario gana también se le notificara.

## Implementación

### Cliente

```
public static void main(String[] args) {  
    Menu menu = new Menu();  
    menu.setVisible(b:true);  
    menu.setLocationRelativeTo(c:null);  
}
```

En nuestro cliente, el método principal es el que se encarga de inicializar la interfaz gráfica donde se le pedirán los datos; se crea una instancia de la clase Menú y con ambos métodos que aparecen en la captura, hacemos que se visualice.

```

public static void enviarDatosAlServidor(String username, String dificultad) {
    try {
        // Conectarse al servidor
        Socket cl = new Socket(host:"localhost", port:3500);
        System.out.println("Conectado al servidor en el puerto: " + cl.getPort());

        // Enviar los datos al servidor
        DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
        dos.writeUTF(username); // Enviar nombre del usuario
        dos.writeUTF(dificultad); // Enviar dificultad
        System.out.println("Datos enviados al servidor: " + username + " con dificultad " + dificultad);

        ObjectInputStream ois = new ObjectInputStream(cl.getInputStream());
        int[][] tablero = (int[][]) ois.readObject();

        //System.out.println(tablero.toString());

        //Juego juego = new Juego(tablero);

        // Cerrar conexiones
        dos.close();
        cl.close();
    } catch(IOException | ClassNotFoundException e) {
        System.out.println("Error al conectar con el servidor: " + e.getMessage());
        e.printStackTrace();
    }
}

```

Contamos con un método el cual se encarga de enviarle los datos necesarios al servidor, para que este se encargue de crear el tablero correcto de acuerdo a la dificultad.



## Servidor

```
public static void main(String[] args) {

    try {
        ServerSocket ss = new ServerSocket(port:3500);
        System.out.println("Servidor iniciado");

        for(;;) {
            //aqui enlazo el client con el server
            Socket cl = ss.accept();
            DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
            DataInputStream dis = new DataInputStream(cl.getInputStream());

            String username = (String) dis.readUTF();
            String difficulty = (String) dis.readUTF();
            System.out.println("Usuario: " + username + " recibido desde " + cl.getInetAddress() + " : " + cl.getLocalPort());
            System.out.println("Con dificultad: " + difficulty);

            //Juego juego = null;
            Tablero tablero = null;
            switch (difficulty) {
                case "facil":
                    //juego = new Juego("facil");
                    tablero = new Tablero(filas:9,columnas:9,minas:10);
                    break;
                case "intermedio":
                    //juego = new Juego("intermedio");
                    tablero = new Tablero(filas:16,columnas:16,minas:40);
                    break;
                case "experto":
                    //juego = new Juego("experto");
                    tablero = new Tablero(filas:16,columnas:30,minas:99);
                    break;
                default:
                    //juego = new Juego("facil");
                    tablero = new Tablero(filas:9,columnas:9,minas:10);
                    break;
            }
            //You, hace 5 días • no gran avance pero necesario (creacion tablero...

            //int[][] tablero = juego.getTablero();

            //ya que tenemos el tablero creado, se lo enviamos al cliente
            ObjectOutputStream oos = new ObjectOutputStream(cl.getOutputStream());
            oos.writeObject(tablero.getTablero());
            oos.flush();

            dis.close();
            dos.close();
            oos.close();
            cl.close();
        }

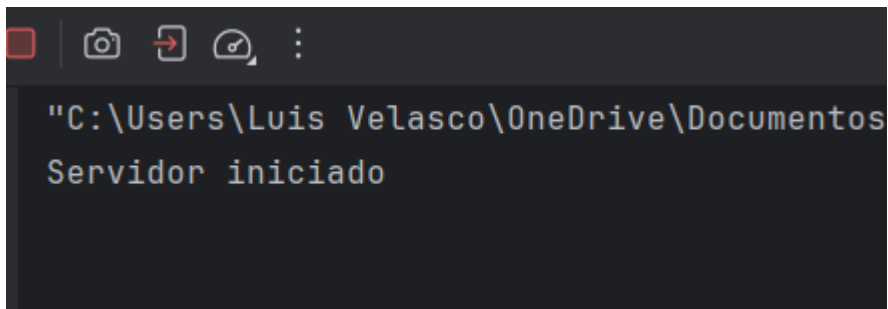
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```

En el servidor, nuestro método principal se encarga de interpretar la dificultad y en base a eso, crear una instancia de nuestra clase Tablero (la cual sirve para crear el tablero) y mandarle las respectivas medidas del tablero.

Toda nuestra lógica del juego la tenemos implementada en clases como Tablero y Juego.

#### Pruebas:

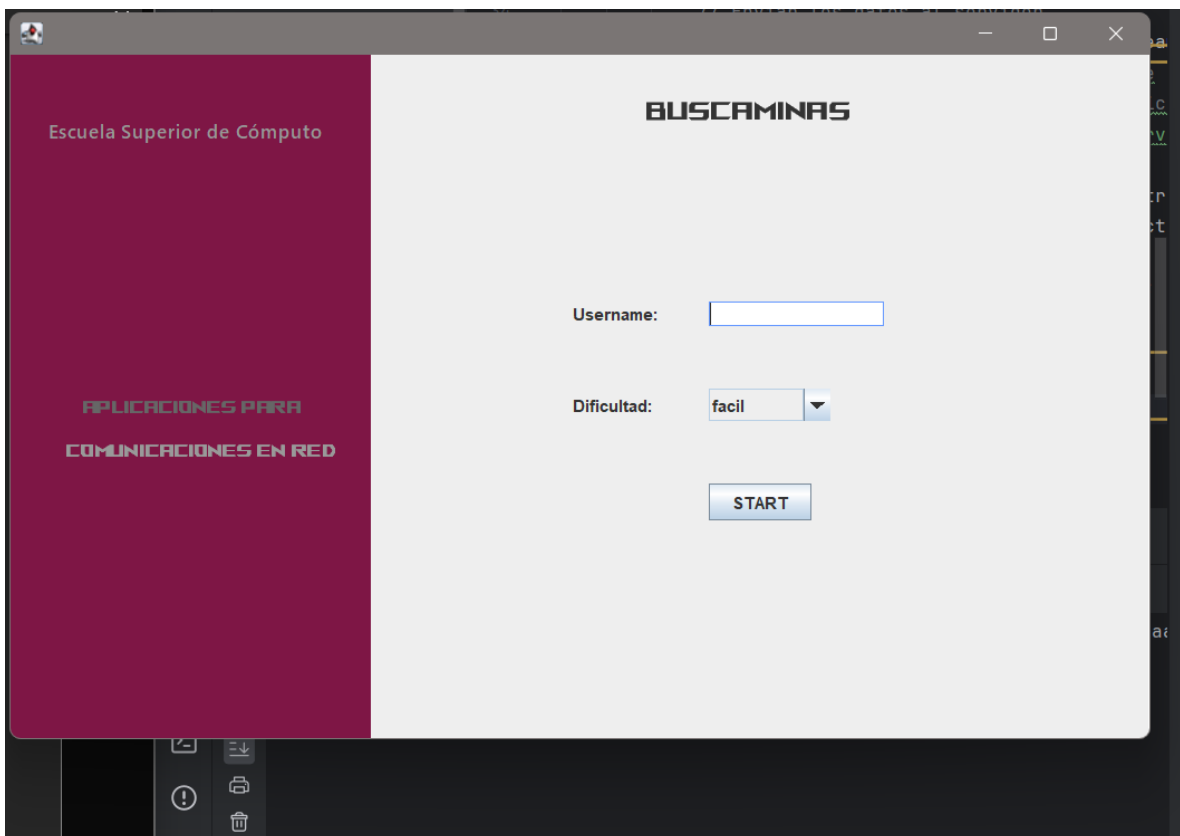
Para que el juego funcione de manera correcta, primero necesitamos ejecutar el servidor para que cliente pueda enlazarse a el.



```
"C:\Users\Luis Velasco\OneDrive\Documentos
Servidor iniciado
```

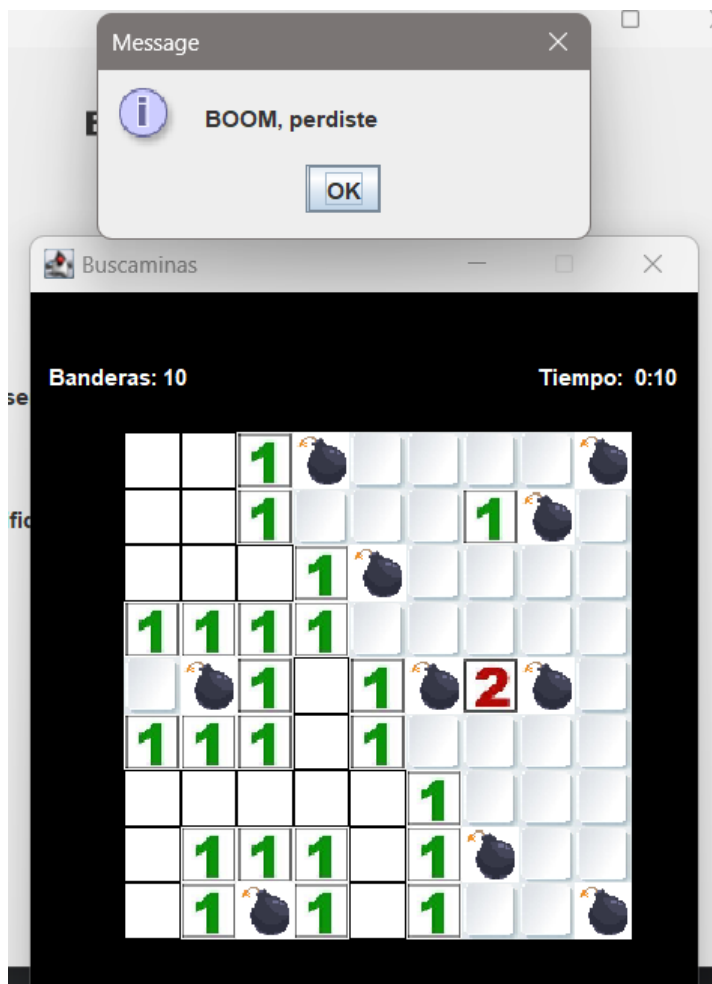
Posteriormente corremos el cliente:

Recordemos que el cliente nos ayuda a visualizar la interfaz menú.



Cabe aclarar que si el usuario no pone su username, el sistema lo considera como “desconocido” ya que no es un campo obligatorio para empezar el juego.

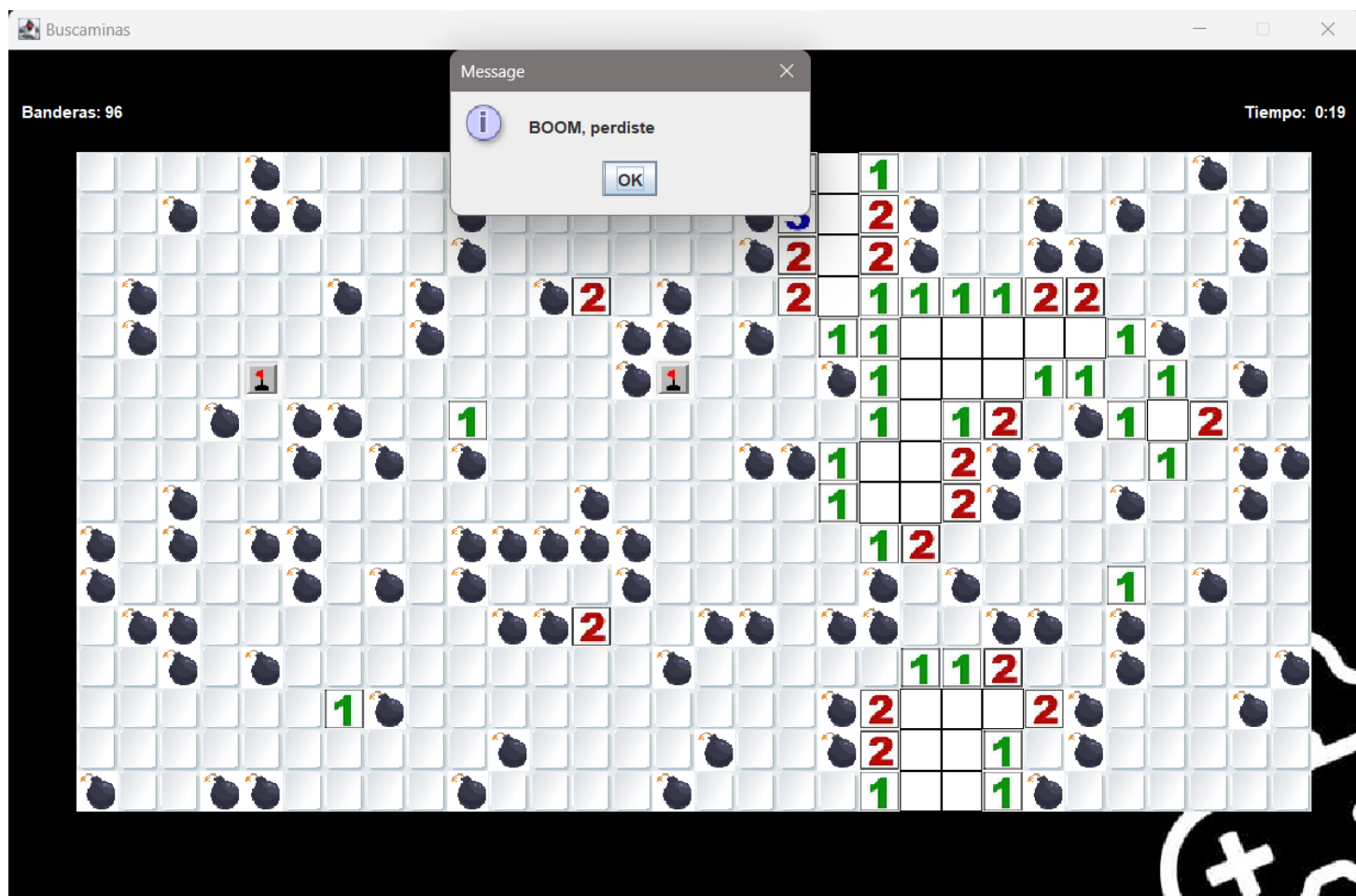
Cuando elegimos la dificultad fácil, se nos despliega el tablero:



Cuando elegimos intermedia:



Cuando elegimos experto:



La consola del servidor nos brinda esto:

```
"C:\Users\Luis Velasco\OneDrive\Documentos\JAVA\jdk-18.0.2\bin\java.exe" -Djava.class.path=.\server.jar -jar server.jar
Servidor iniciado
Usuario: DeSc0n0cId0 recibido desde /127.0.0.1:3500
Con dificultad: facil
Usuario: DeSc0n0cId0 recibido desde /127.0.0.1:3500
Con dificultad: facil
Usuario: pepito recibido desde /127.0.0.1:3500
Con dificultad: intermedio
Usuario: DeSc0n0cId0 recibido desde /127.0.0.1:3500
Con dificultad: intermedio
Usuario: DeSc0n0cId0 recibido desde /127.0.0.1:3500
Con dificultad: experto
|
```

Ya que fue recibiendo uno por uno a los clientes, sin que el servidor se cerrará, que se encuentra en un for infinito.

Mientras que la de la consola del Cliente nos brinda esto:

```
"C:\Users\Luis Velasco\OneDrive\Documentos\JAVA\jdk-18.0.2\bin\java.exe" -Djava.class.path=.\client.jar -jar client.jar
Conectado al servidor en el puerto: 3500
Datos enviados al servidor: DeSc0n0cId0 con dificultad experto
```

## Conclusiones

Flores Morales Aldahir Andrés

En esta práctica, exploramos la implementación de un juego de Buscaminas en Java utilizando una arquitectura cliente-servidor con sockets de flujo bloqueantes. Trabajando con esta tecnología, pude observar cómo los sockets bloqueantes permiten manejar conexiones de manera secuencial, es decir, que el servidor atiende a un cliente a la vez, sin poder aceptar nuevas conexiones hasta que el cliente actual termine su partida. Este enfoque tiene la ventaja de ser relativamente simple de implementar y gestionar, ya que no requiere programación asíncrona ni manejo de eventos complejos.

Sin embargo, esta práctica también me ayudó a reconocer las limitaciones de los sockets bloqueantes en un entorno multijugador. Dado que el servidor está diseñado para atender a los jugadores uno por uno, no es posible que varios jugadores participen simultáneamente, lo cual sería una restricción importante en un juego que busque ofrecer una experiencia interactiva y en tiempo real. A medida que avanza la partida, cualquier cliente nuevo debe esperar a que el anterior termine su juego, lo que puede llevar a demoras y limitar la escalabilidad de la aplicación. En este sentido, la práctica me llevó a reflexionar sobre la necesidad de explorar tecnologías más avanzadas, como sockets no bloqueantes o un enfoque multihilo, que permitan un manejo concurrente de múltiples clientes. Esto sería fundamental para construir una versión de Buscaminas que realmente pueda funcionar en un entorno multijugador sin restricciones de tiempo o de conexión.

Velasco Jimenez Luis Antonio

Durante esta práctica, implementamos el juego de Buscaminas utilizando una arquitectura cliente-servidor con sockets bloqueantes en Java. Este enfoque de comunicación me permitió entender cómo los sockets bloqueantes mantienen una conexión directa y persistente con un cliente, pero al mismo tiempo me hizo ver cómo esto puede ser una desventaja en un entorno donde la concurrencia es importante. En nuestra implementación, el servidor solo puede atender a un jugador a la vez, lo que significa que los jugadores deben esperar su turno para conectarse y jugar. Este flujo secuencial demostró ser una limitación considerable para un juego multijugador, ya que requiere que el servidor finalice una partida antes de recibir una nueva conexión.

Esta práctica me ayudó a comprender tanto los beneficios como las limitaciones de los sockets bloqueantes. Por un lado, son relativamente simples de configurar y proporcionan una conexión estable y directa con el cliente, lo cual resulta útil para juegos de un solo jugador o para aplicaciones donde no se necesita interacción en tiempo real entre múltiples usuarios. Por otro lado, este modelo es claramente insuficiente para juegos con varios jugadores, ya que no permite gestionar conexiones simultáneas, lo que puede llevar a un mal desempeño y a una experiencia de usuario limitada.

Después de completar esta práctica, entiendo mejor la importancia de explorar arquitecturas más avanzadas para escenarios multijugador. Sería interesante considerar la implementación de sockets no bloqueantes o de un servidor multihilo para permitir que el servidor maneje varias conexiones

de manera concurrente. Estos enfoques podrían mejorar la capacidad del juego para soportar múltiples jugadores, brindando una experiencia más fluida y cercana a lo que se espera en un entorno de juego en línea.

### **Bibliografía**

- <https://www.youtube.com/watch?v=3wJTl9LM0sk&t=543s>
- [https://www.youtube.com/watch?v=LdBl0th\\_U\\_Q&t=309s](https://www.youtube.com/watch?v=LdBl0th_U_Q&t=309s)
- <https://parzibyte.me/blog/2018/02/09/sockets-java-chat-cliente-servidor/>
- <https://www.programarya.com/Cursos-Avanzados/Java/Socket>