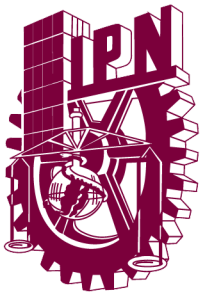

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 2: Sopa de letras

Estudiantes:

Morales Hernández Carlos Jesús

Ramirez Hidalgo Marco Antonio

Grupo: 3CM16

Materia: Aplicaciones para comunicaciones en red

Docente: Axel Ernesto Moreno Cervantes

Carrera: Ingeniería en Sistemas Computacionales

Martes, 15 de marzo del 2022

Índice

Práctica 2: Sopa de letras

1. Introducción

1.1. Modelo Cliente-Servidor

En el modelo cliente-servidor, el dispositivo que solicita información se denomina “cliente”, y el dispositivo que responde a la solicitud se denomina “servidor”. Los procesos de cliente y servidor se consideran parte de la capa de aplicación. El cliente comienza el intercambio solicitando los datos al servidor, quien responde enviando uno o más streams de datos al cliente. Los protocolos de la capa de aplicación describen el formato de las solicitudes y respuestas entre clientes y servidores. Además de la transferencia real de datos, este intercambio también puede requerir la autenticación del usuario y la identificación de un archivo de datos que se vaya a transferir. [1]

Un ejemplo de una red cliente-servidor es el uso del servicio de correo electrónico de un ISP para enviar, recibir y almacenar correo electrónico. El cliente de correo electrónico en una PC doméstica emite una solicitud al servidor de correo electrónico del ISP para que se le envíe todo correo no leído. El servidor responde enviando al cliente el correo electrónico solicitado. [1]

Aunque los datos se describen generalmente como el flujo del servidor al cliente, algunos datos fluyen siempre del cliente al servidor. El flujo de datos puede ser el mismo en ambas direcciones, o inclusive puede ser mayor en la dirección que va del cliente al servidor. Por ejemplo, un cliente puede transferir un archivo al servidor con fines de almacenamiento. Como se muestra en la ilustración, la transferencia de datos de un cliente a un servidor se conoce como “subida” y la transferencia de datos de un servidor a un cliente se conoce como “descarga”. [1]

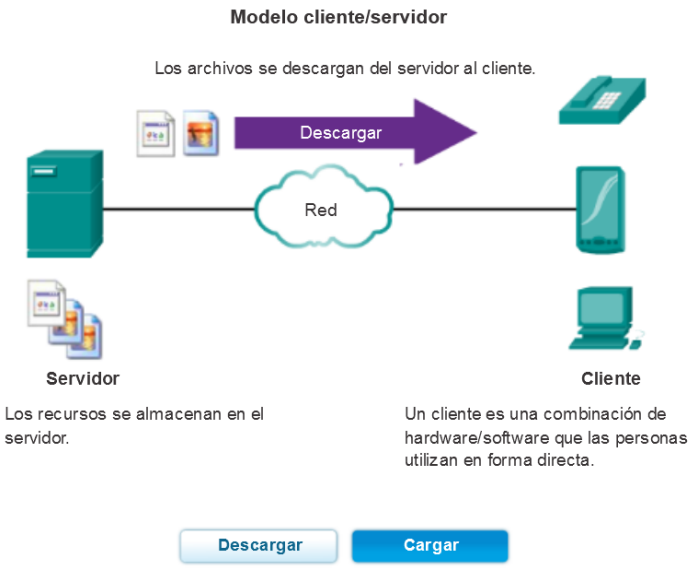

Figura 1. Ilustración del modelo Cliente-Servidor

1.2. Sockets de datagrama

Los sockets de datagramas admiten un flujo de datos bidireccional que no se garantiza que se secuenciará o des duplicará. Tampoco se garantiza que los datagramas sean confiables; pueden no llegar. Es posible que los datos del datagrama lleguen sin orden y posiblemente duplicados, pero se conservan los límites de registro en los datos, siempre y cuando los registros sean menores que el límite de tamaño interno del receptor. [2]

Los datagramas son "sin conexión", es decir, no se establece ninguna conexión explícita; envía un mensaje de datagrama a un socket especificado y puede recibir mensajes de un socket especificado. [2]

Un ejemplo de socket de datagrama es una aplicación que mantiene sincronizados los relojes del sistema en la red. Esto ilustra una funcionalidad adicional de sockets de datagramas en al menos algunas configuraciones: la difusión de mensajes a un gran número de direcciones de red. [2]

Los sockets de datagramas son mejores que los sockets de secuencia para los datos orientados a registros. [2]

2. Desarrollo experimental

Para lograr desarrollar esta práctica se utilizaron tres archivos principales que fueron nombrados *SopaCliente.java*, *SopaServidor.java* y *SopaDeLetras.java*. En seguida, se describen a detalle cada una de estas clases con sus funciones principales que hicieron posible la implementación de la sopa de letras bajo el modelo Cliente-Servidor.

2.1. SopaCliente.java

En esta clase se engloba todo lo correspondiente al cliente, el cual hace peticiones al servidor para recibir los datos para comenzar los juegos y resolver las sopas de letras disponibles.

2.1.1. Función enviarPeticion

Código

```
private void enviarPeticion(String msj) throws Exception {
        InetAddress dst = InetAddress.getByName(this.direccion);
        byte[] b = msj.getBytes();
        DatagramSocket cl = new DatagramSocket();
        this.cl.setBroadcast(true);
        DatagramPacket p = new DatagramPacket(b, b.length, dst, this.puerto);
        this.cl.send(p);
    }
```

Descripción

Esta función se utiliza principalmente para enviar peticiones el servidor, aunque por su naturaleza permite enviar datos de igual manera. Primero, se crea un socket de datagrama y se envía una simple cadena, la cual es interpretada por el servidor para realizar la acción solicitada.

2.1.2. Función

Código

```
public String[] getJuegos() throws Exception {
        // Se piden los juegos
        this.enviarPeticion("getJuegos");

        // Se reciben los juegos
        DatagramPacket p = new DatagramPacket(new byte[65535], 65535);
        this.cl.receive(p);
        ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(p.getData()));
        String[] juegos = (String[]) ois.readObject();

        return juegos;
    }
```

Descripción

Una vez se ha enviado una petición al servidor para obtener todos los juegos disponibles, el cliente debe esperar la respuesta con los datos enviados desde el servidor. Esto es lo que hace esta función, a la cual llegan todos y cada uno de los juegos con los que se cuenta en el momento de hacer la petición. De igual manera, se hace con un socket de datagrama que espera por un objeto, dicho objeto no es nada más ni menos que un arreglo de cadenas que describen el concepto de la sopa de letras.

2.1.3. Función obtenerPalabras

Código

```
public String[] obtenerPalabras() throws Exception {

        // Se piden los conceptos
        this.enviarPeticion("juego-" + this.juego);

        // Se reciben los conceptos
        DatagramPacket p = new DatagramPacket(new byte[65535], 65535);
        this.cl.receive(p);
        ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(p.getData()));
        String[] conceptos = (String[]) ois.readObject();

        return conceptos;
    }
```

Descripción

Esta función hace lo mismo que la anterior, pero después de una petición para obtener todas las palabras relacionadas a un mismo concepto. Estas son las palabras que se le mostrarán al usuario en la sopa de letras.

2.1.4. Función jugar

Código

```
public void jugar() throws Exception{
        // Se crea la sopa de letras
```

```
        SopaDeLetras sdl = new SopaDeLetras(this.juego, this.concepto, this.definicion);

        // Se toma el inicio del tiempo para jugar
        long tiempoInicial = System.nanoTime();
        String estats = sdl.comenzarJuego();

        long tiempoFinal = System.nanoTime();
        this.tiempo = (tiempoFinal - tiempoInicial) / 1000000000;

        // Se sacan las estadisticas del juego reciente terminado
        String[] juegoReciente = estats.split("-");

        this.encontradas = Integer.parseInt(juegoReciente[0]);
        this.aEncontrar = Integer.parseInt(juegoReciente[1]);
        this.terminado = Boolean.parseBoolean(juegoReciente[2]);
    }
```

Descripción

La función jugar es la que comienza la ejecución del juego y recopila los datos requeridos para hacer las estadísticas de cada juego por concepto. En esta función, lo primero que se hace es crear e inicializar el objeto que representa la sopa de letras. Asimismo, se lleva registro del tiempo que le toma al usuario jugar con esa sopa de letras. Aquí no se hacen peticiones al servidor pues ya se tienen todos los datos necesarios,

2.1.5. Función guardarEstadisticas

Código

```
public void guardarEstadisticas() throws Exception {
        limpiarPantalla();
        System.out.println("G U A R D A   T U S   E S T A D I S T I C A S\n");

        System.out.print("Dale un nombre a estas estadisticas: ");
        Scanner teclado = new Scanner(System.in);
        String nombre = teclado.nextLine();

        String estats = this.juego + "-";
        estats += nombre + "-";
        estats += this.tiempo + "-";
        estats += this.terminado + "-";
        estats += this.encontradas + "-";
        estats += this.aEncontrar;

        this.enviarPeticion(estats);
    }
```

Descripción

Esta función recopila todas y cada una de las estadísticas generadas por una sesión de juego con alguna sopa de letras. Todos los datos se reúnen en una misma cadena, y aprovechando la función *enviarPeticion* que envía una cadena al servidor, se envía una petición de guardar las estadísticas, por medio de una petición a la que se le añaden todos los datos a guardar.

2.1.6. Función obtenerEstadisticas

Código

```
public String[] obtenerEstadisticas() throws Exception {

        // Se hace la peticion de las estadisticas
        this.enviarPeticion("estats-" + this.juego);

        // Se obtienen las estats
        DatagramPacket p = new DatagramPacket(new byte[65535], 65535);
        this.cl.receive(p);
        ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(p.getData()));
        String[] estats = (String[]) ois.readObject();

        return estats;
    }
```

Descripción

Así como todas las otras funciones que hacen peticiones al servidor por medio de sockets de datagrama, esta espera como respuesta un arreglo de cadenas que tienen las estadísticas de todos y cada uno de los juegos guardados en el servidor hasta el momento.

2.1.6. Función iniciarJuego

Código

```
public void iniciarJuego() throws Exception {
        limpiarPantalla();

        String[] juegos = this.getJuegos();
        this.setJuego(this.elegirJuego(juegos));
        String[] conceptos = this.obtenerPalabras();
        this.descomponerConceptos(conceptos);
        this.jugar();
        this.guardarEstadisticas();
        String[] estats = this.obtenerEstadisticas();
        this.mostrarEstadisticas(estats);
    }
```

Descripción

Esta función tiene la responsabilidad de ejecutar cada una de las funciones descritas hasta el momento para esta clase en el orden requerido.

2.2. Archivo SopaServidor.java

En este archivo se almacenan cada una de las funciones relacionadas al servidor del juego, así como la clase que las engloba todas. El servidor en sí traba en la dirección reservada como localhost y se habilita para hacer una difusión de broadcast para todos los clientes disponibles.

2.2.1. Función iniciarServidor

Código

```
private void iniciarServidor() throws Exception{
        while(true){
            this.s.receive(this.p);
            String peticion = new String(this.p.getData(), 0, this.p.getLength());
            this.procesarPeticion(peticion);
        }
    }
```

Descripción

La función *iniciarServidor* se encarga de poner al servidor a recibir datagramas por medio de un socket de datagramas provenientes del cliente, todo en un ciclo infinito. Ya que se tienen estos datagramas, se procesa la solicitud que ha hecho el cliente.

2.2.2. Función procesarPeticion

Código

```
private void procesarPeticion(String peticion) throws Exception{
        // Se pide la lista de juegos
        if(peticion.equals("getJuegos")){
            String[] juegos = this.getJuegos();
            System.out.println("Se ha recibido una peticion para consultar las sopas disponibles de
" + p.getAddress() + "...");
            this.enviarJuegos(juegos);
            System.out.println("Se enviaron los juegos disponibles a " + p.getAddress() + "...");
        }
        // Se pide un juego
        else if (peticion.contains("juego-")){
            String juego = peticion.substring(6);
            System.out.println("Se ha recibido una peticion para enviar los conceptos y definiciones
del juego " + juego + " de " + p.getAddress() + "...");
            String[] conceptos = this.leerArchivoJuego(juego);
            this.enviarConceptos(conceptos);
            System.out.println("Se enviaron los conceptos y definiciones del juego " + juego + " a "
+ p.getAddress() + "...");
        }
        // Se mandan las estadisticas
        else if (peticion.contains("estats-")){
            String juego = peticion.substring(7);
            System.out.println("Se ha recibido una peticion para enviar las estdisticas del juego "
+ juego + " de " + p.getAddress() + "...");
            String[] estats = this.obtenerEstats(juego);
            this.enviarEstats(estats);
            System.out.println("Se enviaron las estadisticas del juego " + juego + " a " +
p.getAddress() + "...");
        }
        // Se envian las estadisticas
        else {
            System.out.println("Se ha recibido una peticion para guardar las estadisticas de un
juego de " + p.getAddress() + "...");
            String[] estats = peticion.split("-");
            this.guardarEstadisticas(estats);
            System.out.println("Se guardaron las estadisticas del juego de " + p.getAddress() +
"...");
```

```
        }
    }
```

Descripción

Esta función es larga porque además de interpretar el mensaje del cliente, el servidor informa al usuario lo que se está pidiendo, de dónde viene y si la petición se ha enviado de vuelta al cliente con éxito. Se tienen cuatro peticiones: para obtener los juegos disponibles, para obtener las palabras de un juego, para enviar las estadísticas al cliente y, finalmente, para guardar las estadísticas de un juego.

2.2.3. Función iniciarServidor

Código

```
private void iniciarServidor() throws Exception{
        while(true){
            this.s.receive(this.p);
            String peticion = new String(this.p.getData(), 0, this.p.getLength());
            this.procesarPeticion(peticion);
        }
    }
```

Descripción

La función *iniciarServidor* se encarga de poner al servidor a recibir datagramas por medio de un socket de datagramas provenientes del cliente, todo en un ciclo infinito. Ya que se tienen estos datagramas, se procesa la solicitud que ha hecho el cliente.

2.2.4. Función getJuegos

Código

```
private String[] getJuegos(){
        File carpeta = new File(this.rutaJuegos);
        return carpeta.list();
    }
```

Descripción

Esta función lee todos los folders que se encuentran en el folder donde se almacenan todos los juegos. Cada folder dentro de este último mencionado contiene todo lo referente a un mismo juego y describe por sí mismo la disponibilidad de los juegos el utilizarse.

2.2.5. Función getJuegos

Código

```
private void enviarJuegos(String[] juegos) throws Exception{
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
```

```
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(juegos);
        oos.flush();
        byte[] b = baos.toByteArray();
        DatagramPacket p = new DatagramPacket(b, b.length, this.p.getAddress(), this.p.getPort());
        this.s.send(p);
    }
```

Descripción

Esta función responde al cliente con un arreglo de cadenas que son referentes a los juegos disponibles al momento de mandarse a llamar. Esto se hace por medio de un paquete de datagrama. Además, el cliente debe estar a la respuesta del servidor en dicho instante.

2.2.6. Función leerArchivoJuego

Código

```
public String[] leerArchivoJuego(String juego) throws Exception {
        String rutaArchivo = rutaJuegos + "/" + juego + "/";
        String nombreArchivo = juego + ".txt";
        List<String> conceptos = new ArrayList<String>();

        File juegoArchivo = new File(rutaArchivo + nombreArchivo);
        Scanner lector = new Scanner(juegoArchivo);

        while (lector.hasNextLine()){
            conceptos.add(lector.nextLine());
        }

        return conceptos.toArray(new String[0]);
    }
```

Descripción

Con esta función se abre un archivo de un juego que contiene cada una de las palabras que se pondrán en la sopa de letras. Estas palabras se almacenan en un arreglo de cadenas que se enviará al cliente posteriormente.

2.2.7. Función

Código

```
private void enviarConceptos(String[] conceptos) throws Exception{
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(conceptos);
        oos.flush();
        byte[] b = baos.toByteArray();
        DatagramPacket p = new DatagramPacket(b, b.length, this.p.getAddress(), this.p.getPort());
        this.s.send(p);
    }
```

Descripción

Esta función se utiliza tan pronto como se termina de usar la anterior, pues envía las palabras que se colocarán en la sopa de letras por medio de un paquete de datagrama.

2.2.8. Función obtenerEstats

Código

```
private String[] obtenerEstats(String juego) throws Exception{
        List<String> estatsTodos = new ArrayList<String>();
        String estats = "";
        File estatsArchivo;
        Scanner lector;

        String ruta = rutaJuegos + "/" + juego + "/estadisticas/";
        File carpeta = new File(ruta);
        String[] archivos = carpeta.list();

        for(String archivo : archivos){
            estatsArchivo = new File(ruta + archivo);
            lector = new Scanner(estatsArchivo);

            while (lector.hasNextLine()){
                estats += lector.nextLine();

                if(lector.hasNextLine()){
                    estats += "-";
                }
            }

            //estatsArchivo.close();
            estatsTodos.add(estats);
            estats = "";
        }

        return estatsTodos.toArray(new String[0]);
    }
```

Descripción

Esta función lee todos los datos que almacenan archivos en los que se guardan las estadísticas de todos los juegos en los que ha sido participe el usuario. Los datos se formatean en una cadena que será interpretada por el cliente cuando este reciba el datagrama correspondiente.

2.2.9. Función enviarEstats

Código

```
private void enviarEstats(String[] estats) throws Exception {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(estats);
        oos.flush();
        byte[] b = baos.toByteArray();
        DatagramPacket p = new DatagramPacket(b, b.length, this.p.getAddress(), this.p.getPort());
        this.s.send(p);
    }
```

Descripción

Esta función envía las estadísticas, que se obtienen con la función descrita anteriormente, al cliente. El proceso es el mismo que con todas las demás funciones, se envía un arreglo de cadenas.

2.2.8. Función guardarEstadisticas

Código

```
private void guardarEstadisticas(String[] estats) throws Exception {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd-HHmmss");
        String horario = dtf.format(LocalDateTime.now());
        String carpeta = rutaJuegos + "/" + estats[0] + "/estadisticas/";

        File file = new File(carpeta + estats[1] + horario + ".txt");

        FileWriter fw = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(estats[1]);
        bw.newLine();
        bw.write(estats[2]);
        bw.newLine();
        bw.write(estats[3]);
        bw.newLine();
        bw.write(estats[4]);
        bw.newLine();
        bw.write(estats[5]);
        bw.close();
    }
```

Descripción

Esta función se ejecuta una vez se recibe una petición para guardar las estadísticas de un juego desde un cliente. En sí, estos datos se formatean de tal manera que sea fácil leerlos cuando sea necesario y se guardan en un archivo de texto que se almacena en un folder correspondiente a las estadísticas de un juego.

2.3. Archivo SopaDeLetras.java

Como la sopa de letras es independiente del servidor y el cliente quienes se encargan de la comunicación, se realizó una tercera clase que contiene todo lo referente al juego de la sopa de letras. Esta clase se emplea únicamente del lado del cliente y los datos con los que se inicializa provienen del servidor.

2.3.1. Constructor SopaDeLetras

Código

```
SopaDeLetras(String concepto, String[] conceptos, String[] definiciones){
        this.concepto = concepto;
        this.conceptos = conceptos;
        this.definiciones = definiciones;
```

```
        this.totalPalabras = conceptos.length;

        this.terminado = false;
        this.totalEncontradas = 0;
        tablero = new String [this.LONGITUD][this.LONGITUD];
        ocupacion = new Boolean [this.LONGITUD][this.LONGITUD];

        for(int i = 0; i < this.LONGITUD; i++){
            for(int j = 0; j < this.LONGITUD; j++){
                tablero[i][j] = "-";
                ocupacion[i][j] = false;
            }
        }

        palabras = new HashMap <> ();
        palabrasEncontradas = new ArrayList <String>();
        direcciones = new HashMap <> ();
        direcciones.put(0, new Direccion("NORTE", 0, 1));
        direcciones.put(1, new Direccion("NORESTE", 1, 1));
        direcciones.put(2, new Direccion("ESTE", 1, 0));
        direcciones.put(3, new Direccion("SURESTE", 1, -1));
        direcciones.put(4, new Direccion("SUR", 0, -1));
        direcciones.put(5, new Direccion("SUROESTE", -1, -1));
        direcciones.put(6, new Direccion("OESTE", -1, 0));
        direcciones.put(7, new Direccion("NOROESTE", -1, 1));

        this.construirTablero();
    }
```

Descripción

El constructor de esta clase recibe los datos del servidor e inicializa todos los objetos y variables de los que se auxiliara para implementar el juego de la sopa de letras.

2.3.2. Función constuirTablero

Código

```
public void construirTablero(){
        for(String concepto : this.conceptos){
            this.colocarPalabra(concepto);
        }

        // Se termina de rellenar la tabla
        for(int i = 0; i < this.LONGITUD; i++){
            for(int j = 0; j < this.LONGITUD; j++){
                if(!ocupacion[i][j]){
                    int assci = randomInt(97, 122);
                    char relleno = (char) assci;
                    tablero[i][j] = Character.toString(relleno);
                }
            }
        }
    }
```

Descripción

Esta función es corta, pues para construir el tablero solo se necesita de colocar todas las palabras en este y terminarlo de rellenar con letras aleatorias para así conseguir una verdadera sopa de letras. Lo divertido aquí es cómo se colocan esas palabras.

2.3.3. Función colocarPalabra

Código

```
private void colocarPalabra(String concepto){
        Boolean posicionCorrecta = false;
        int minX, maxX, minY, maxY;
        int longitud = concepto.length();
        Direccion direccion = null;
        Coordenadas inicio = new Coordenadas();

        minX = minY = 0;
        maxX = maxY = this.LONGITUD - 1;

        while(!posicionCorrecta){

            // Se supone que se genera una posicion correcta
            posicionCorrecta = true;
            direccion = this.direcciones.get(randomInt(0, 7));
            inicio = inicio.generarCoordenadas(minX, maxX, minY, maxY);

            // Revisar correcto solapamiento
            for(int i = 0; i < longitud; i++){
                int movX = inicio.x + (direccion.movX * i);
                int movY = inicio.y + (direccion.movY * i);

                if(movX >= this.LONGITUD || movX < 0){
                    posicionCorrecta = false;
                    break;
                }

                if(movY >= this.LONGITUD || movY < 0){
                    posicionCorrecta = false;
                    break;
                }

                if(ocupacion[movX][movY]){
                    if(!tablero[movX][movY].equals(concepto.substring(i, i + 1))){
                        posicionCorrecta = false;
                        break;
                    }
                }
            }

            // Se revisa que no haya una misma palabra con las misma coordenadas de inicio y misma
direccion
            for (Map.Entry <String, Palabra> palabra : palabras.entrySet()) {
                Palabra guardada = palabra.getValue();

                if(direccion.mismaDireccion(guardada.direccion) &&
inicio.mismasCoords(guardada.coordsInicio)){
                    posicionCorrecta = false;
                    break;
                }
            }
        }
```

```
        // Se ocupa el tablero
        int movX = 0;
        int movY = 0;

        for(int i = 0; i < longitud; i++){
            movX = inicio.x + (direccion.movX * i);
            movY = inicio.y + (direccion.movY * i);

            ocupacion[movX][movY] = true;
            tablero[movX][movY] = concepto.substring(i, i + 1);
        }

        // Se guarda la palabra
        int coordX = movX;
        int coordY = movY;
        Coordenadas coordsFinal = new Coordenadas(coordX, coordY);
        palabras.put(concepto, new Palabra(concepto, direccion, inicio, coordsFinal));
    }
```

Descripción

Esta función es el pilar del juego, pues coloca las palabras con las ocho direcciones especificadas en el planteamiento de la práctica. Además de que permite que las palabras se crucen entre sí para conseguir una sopa todavía más apegada a la que todos conocemos desde pequeños.

Primero, se genera una posición inicial aleatoria para la palabra, después, se genera una dirección que tendrá esa palabra en la sopa. Ya que se tiene esto, se determina si la dirección y la posición inicial son correctas. En caso de que esto sea afirmativo, se revisa que la palabra no solape el contenido de otra. Finalmente, si todo sale correctamente, se guarda esta palabra y se escribe en la sopa, que se representa con una matriz de cadenas.

2.3.4. Función comenzarJuego

Código

```
public String comenzarJuego() throws Exception {
        int opc = 1;
        Scanner teclado;
        String palabra;
        int inicioX, inicioY, finalX, finalY;
        String estats = "";

        while(opc != 2){
            limpiarPantalla();
            this.imprimirTablero();

            System.out.println("\nOpciones");
            System.out.println("\t1. Descubrir palabra");
            System.out.println("\t2. Salir");
            System.out.print("Elegir opcion: ");

            teclado = new Scanner(System.in);
            opc = teclado.nextInt();

            switch (opc){
                case 1:
                    System.out.print("\nEscribe la palabra: ");
                    teclado = new Scanner(System.in);
```

```
                    palabra = teclado.nextLine();

                    System.out.print("Escribe la coordenada x de inicio: ");
                    teclado = new Scanner(System.in);
                    inicioX = teclado.nextInt();

                    System.out.print("Escribe la coordenada y de inicio: ");
                    teclado = new Scanner(System.in);
                    inicioY = teclado.nextInt();

                    System.out.print("Escribe la coordenada x de final: ");
                    teclado = new Scanner(System.in);
                    finalX = teclado.nextInt();

                    System.out.print("Escribe la coordenada y de final: ");
                    teclado = new Scanner(System.in);
                    finalY = teclado.nextInt();

                    this.procesarEleccion(palabra, inicioY, inicioX, finalY, finalX);

                    if(this.comprobarFin()){
                        limpiarPantalla();
                        System.out.println("\nHAS GANADO!!!!\n");
                        opc = 2;
                        pausa("Presiona una tecla para continuar... ");
                    }

                    break;
                case 2:
                    break;
            }
        }
        estats += this.totalEncontradas + "-";
        estats += this.totalPalabras + "-";
        estats += this.terminado;

        return estats;
    }
```

Descripción

Esta función se debe llamar siempre después de que se construye el tablero, no antes. Lo que permite es que el usuario pueda interactuar con el tablero de la sopa de letras y señalar qué palabras ha sido capaz de encontrar. Asimismo, reconoce el momento en el que se llega a un estado de victoria y el juego se puede finalizar.

2.3.5. Función procesarEleccion

Código

```
private void procesarEleccion(String palabra, int inicioX, int inicioY, int finalX, int finalY){
        if(this.eleccionValida(palabra, inicioX, inicioY, finalX, finalY)){

            // Se guarda la palabra como encontrada
            palabrasEncontradas.add(palabra);

            // Se extrae palabra para obtener movimiento
            Palabra p = palabras.get(palabra);

            // "\033[0;101m"
```

```
            for(int i = 0; i < palabra.length(); i++){
                int movX = p.coordsInicio.x + (p.direccion.movX * i);
                int movY = p.coordsInicio.y + (p.direccion.movY * i);

                tablero[movX][movY] = "\033[0;101m" + tablero[movX][movY] + "\033[0m";
            }

            System.out.println("\nCorrecto!!!!\n");
            pausa("Presiona una tecla para continuar... ");
        }
        else {
            System.out.println("\nEstas equivocado :((\n");
            pausa("Presiona una tecla para continuar... ");
        }
    }
```

Descripción

Con esta función se determina si los datos que han ingresado el usuario corresponden a una selección de palabra valida. En caso de que sea así, la palabra se selecciona de otro color en el tablero. Si esto no ocurre, simplemente se le avisa al usuario que lo que ha hecho no es correcto y se le da la oportunidad de intentarlo de nuevo.

3. Conclusiones

En esta práctica se trabajaron con los sockets de datagrama para implementar el juego de la sopa de letras. Como se hizo en clase, se enviaron en todas las ocasiones paquetes que contenían objetos, estos objetos se decidieron manejar como cadenas o arreglos de cadenas.

El cliente que se implementó envía peticiones el servidor, estas peticiones son cadenas que están configuradas de tal forma que para el servidor sea fácil interpretar lo que el cliente esté requisitando. Esto se hace así para evitar el envío de datagramas de gran tamaño y propiciar menos la pérdida de datos o la duplicidad.

Cada que el cliente hacer una petición, también espera en seguida la respuesta del servidor con los datos que se están pidiendo. El servidor siempre está a la escucha de esto y cuando detecta que un cliente quiere algo, se interpreta lo que está pidiendo y se envía lo que el servidor tiene disponible en este momento.

Tanto el cliente como el servidor se encargan únicamente de la comunicación de los datos, peticiones y respuestas. Sin embargo, esto es independiente del juego de sopa de letras, por lo que se tuvo la necesidad de crear un objeto que se encargue específicamente de lo que implica el juego.

Así, se logró implementar un juego de sopa de letras en la modalidad de concepto totalmente funcional que funciona bajo el modelo Cliente-Servidor. Esta sopa de letra se ejecuta únicamente del lado del cliente, pues no es necesario que el servidor intermedie en la funcionalidad del juego ya que todos los datos se pueden enviar independientemente de si se juega o no, aunque al final sí se necesita recopilar los datos refrentes a las estadísticas.

Bibliografía

[1] Instituto Tecnológico de Roque. *Protocolos de la capa de aplicación*. [En línea]. Disponible en: http://itroque.edu.mx/cisco/cisco1/course/module10/10.1.2.5/10.1.2.5.html

[2] Microsoft. (2022, marzo 12). *Windows Sockets: Sockets de datagramas*. [En línea]. Disponible en: https://docs.microsoft.com/es-es/cpp/mfc/windows-sockets-datagram-sockets?view=msvc-170