INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

----------**APLICACIONES EN COMUNICACIONES EN RED**----------

PRÁCTICA 2:

Sopa de letras usando sockets de datagrama

Alumno:

Meza Vargas Brandon David

Grupo:
3CM16

Profesor:

Moreno Cervantes Axel Ernesto

índice

índice de Ilustraciones

Introducción

Los sockets de datagrama son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En el presente reporte se detallará la práctica realizada haciendo uso de sockets de datagrama para poder enviar datos de un cliente a un servidor y viceversa con el objetivo de hacer un juego de sopa de letras.

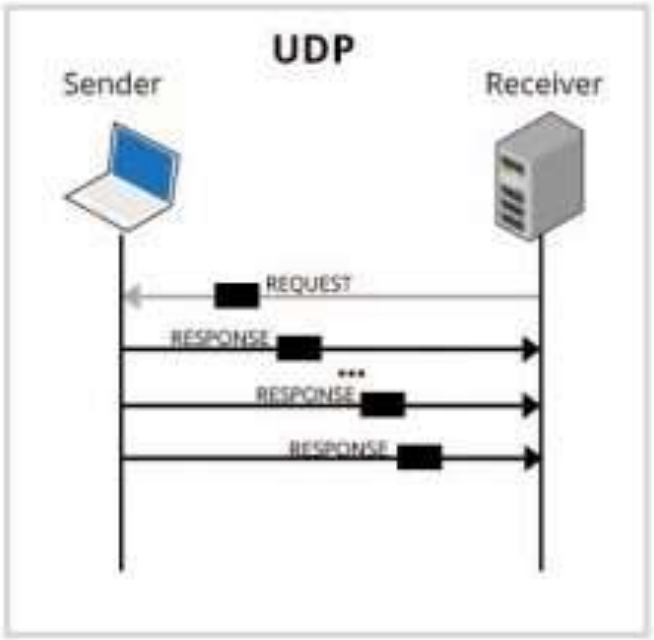


Ilustración 1. Sockets de datagrama.

Desarrollo

A continuación se muestra una explicación de cada parte que conforma la sopa de letras creada en esta práctica, así como las capturas correspondientes a cada método creado.

Sopa de letras

Método setSoup

Este método se encarga de setear la sopa de letras, establece el límite de filas y columnas, además de que va estableciendo la orientación de las palabras de manera aleatoria.

De igual forma se establece el espacio necesario para que las palabras quepan de manera diagonal. Dependiendo del tamaño de la palabra y el espacio disponible se acomodará la palabra en una determinada posición, verificando antes que la palabra no choque con alguna otra palabra ya acomodada en la sopa. En caso de que no choquen, se acomoda la palabra en una de 8 diferentes posiciones: vertical, vertical invertida, horizontal, horizontal invertido, diagonal derecha hacia arriba, diagonal derecha hacia abajo, diagonal izquierda hacia arriba y diagonal izquierda hacia abajo.

En caso de que una palabra choque se le asignará otra orientación para poder acomodarla. En la siguiente ilustración podemos ver el código de este método.

```java
/**
 * @brief Method that set all the soup
 */
public void setSoup(){

    for(int i = 0; i < words.length; i++){
        String currentWord = words[i];
        rand = new Random();

        Integer limitRow = 15;
        Integer limitColumn = 15;
        //*Word long
        Integer letters = currentWord.length();
        //*Random word position
        Integer rowPosition = rand.nextInt(limitRow);
        Integer columnPosition = rand.nextInt(limitColumn);

        //*Checking if space is enough
        Integer aCol = (limitColumn ) - columnPosition;
        Integer aRow = (limitRow ) - rowPosition;

        Integer aDRow;
        Integer aLRow;
        Integer aDRDRow;
        Integer aDLDRow;

        //*Stablishing rightDiagonal space
        aDRow = (aCol < (rowPosition+1) ) ? aCol : rowPosition+1;
        if(aCol == rowPosition+1){
            aDRow = aCol;
        }
        //*Stablishing leftDiagonal space
        aLRow = (( columnPosition + 1 ) < ( rowPosition + 1 ) ? columnPosition + 1 : rowPosition + 1);
        if( columnPosition + 1 == rowPosition + 1){
            aLRow = columnPosition + 1;
        }

        //*Stablishing right Diagonal down space
        aDRDRow = ( aCol < aRow ) ? aCol : aRow;
        if(aCol == aRow){
            aDRDRow = aCol;
        }
```

```
//*Stablishing left Diagonal down space
aDLDRow = ( columnPosition + 1 < aRow ) ? columnPosition + 1 : aRow;
if( columnPosition + 1 == aRow ){
    aDLDRow = columnPosition + 1;
}

Integer fillAction = 8; //*To generate random word orientation
Integer randomAction = rand.nextInt(fillAction);     //*Generating random number between 0 - 7

  Integer action = setOrientation(randomAction);
//*If that verifies the orientation and if there's space enouth to place the word in that orientation
if(
    (aRow >= letters && randomAction == 0) || (aCol >= letters && randomAction == 1) || (rowPosition+1 >= letters && randomAction == 2)
    || (columnPosition+1 >= letters && randomAction == 3) || ( aDRow >= letters && randomAction == 4 ) || ( aLRow >= letters && randomAction == 5 )
    || ( aDRDRow >= letters && randomAction == 6 ) || ( aDLDRow >= letters && randomAction == 7 )

 ){
   if(!isWordCollapsed(randomAction, columnPosition, rowPosition, letters)){

       switch(randomAction){
           case 0:
               vertical(letters, rowPosition, columnPosition, currentWord);
               break;
           case 1:
               horizontal(letters, rowPosition, columnPosition, currentWord);
               break;
           case 2:
               invertedVertical(letters, rowPosition, columnPosition, currentWord);
               break;
           case 3:
               invertedHorizontal(letters, rowPosition, columnPosition, currentWord);
               break;
           case 4:
               rightDiagonal(letters, rowPosition, columnPosition, currentWord);
               break;
           case 5:
               leftDiagonal(letters, rowPosition, columnPosition, currentWord);
               break;
           case 6:
               rightDownDiagonal(letters, rowPosition, columnPosition, currentWord);
               break;
           case 7:
               leftDownDiagonal(letters, rowPosition, columnPosition, currentWord);
               break;

                       default:
                           System.err.println("Not valid orientation");
                           break;
                   }

               } else i--; //*Taking other position if there's collision
           } else i--; //*Taking other positions if there's no enough space for place the word

       }
   }
```

Ilustración 2. Método setSoup.

Método vertical

Este método acomoda la palabra de manera vertical recorriendo las filas desde la posición que se le asigno de manera automática a la palabra.

```java
/**
 * Method that places a word in the vertical orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void vertical(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0; i < lWord; i++){
        soup[rowPos+i][colPos] = "" + word.charAt(i);
          rowPos++;
    }
}
```

Ilustración 3. Método vertical

Método horizontal

Este método acomoda la palabra de manera horizontal recorriendo las columnas desde la posición que se le asigno de manera aleatoria a la palabra.

```java
/**
 * Method that places a word in the horizontal orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void horizontal(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0; i < lWord; i++){
        soup[rowPos][colPos+i] = ""+word.charAt(i);
    }
}
```

Ilustración 4. Método horizontal

Método invertedVertical

Este método acomoda la palabra en la posición vertical invertida, recorriendo las filas y disminuyendo la posición de fila de la palabra.

```java
/**
 * Method that places a word in the inverted verrtical orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void invertedVertical(Integer lWord, Integer rowPos, Integer colPos, String word){
   for(int i = 0; i < lWord; i++){
       soup[rowPos-i][colPos] = ""+word.charAt(i);
   }
}
```

Ilustración 5. Método invertedVertical

Método invertedHorizontal

Este método acomoda la palabra de manera horizontal invertido recorriendo las columnas y disminuyendo la columna desde la posición que se le asigno a la palabra de manera aleatoria.

```
/**
 * Method that places a word in the inverted horizontal orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void invertedHorizontal(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0 ; i < lWord; i++){
        soup[rowPos][colPos-i] = ""+word.charAt(i);
    }
}
```

Ilustración 6. Método invertedHorizontal

Método rightDiagonal

Este método acomoda la palabra en diagonal derecha hacia arriba, recorriendo la posición de la fila y columna

```
/**
 * Method that places a word in the right diagonal orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void rightDiagonal(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0; i < lWord; i++){
        soup[rowPos-i][colPos+i] = ""+word.charAt(i);
    }
}
```

Ilustración 7. Método rightDiagonal

Los siguientes métodos son similares al anterior, solo se aumenta o disminuye la posición de la columna o fila dependiendo si es hacia arriba o abajo o izquierda o derecha, por esto solo se incluirán las capturas.

Método leftDiagonal

```
/**
 * Method that places a word in the left diagonal orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void leftDiagonal(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0; i < lWord; i++){
        soup[rowPos-i][colPos-i] = "" + word.charAt(i);
    }
}
```

Ilustración 8. Método leftDiagonal

Método rightDownDiagonal

```
/**
 * Method that places a word in the right down vertical orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void rightDownDiagonal(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0; i < lWord; i++){
        soup[rowPos+i][colPos+i] = "" + word.charAt(i);
    }
}
```

Ilustración 9. Método rightDownDiagonal

Método leftDownDiagonal

```
/**
 * Method that places a word in the left down diagonal orientation inside the soup
 * @param {Integer} lWord the word longitude
 * @param {Integer} rowPos the row position in the soup
 * @param {Integer} colPos the column position un the soup
 * @param {String} word  The word to place in
 */
public void leftDownDiagonal(Integer lWord, Integer rowPos, Integer colPos, String word){
    for(int i = 0; i < lWord; i++){
        soup[rowPos+i][colPos-i] = "" + word.charAt(i);
    }
}
```

Ilustración 10. Método leftDownDiagonal

Método isWordCollapsed

Este método indica si una palabra esta colapasando con otra dentro de la sopa. Dependiendo de la orientación de la palabra va checando si en la posición que se esta colocando ya hay escrito algún otro carácter que haga que colapse y por lo tanto no se pueda colocar la palabra completa.

Si no choca con nada regresa false pues si se podrá colocar esta palabra.

```
/**
 * @brief Method that stablish if a word is collapsed or not inside the soup
 * @param {Integer} orientation the word orientation
 * @param {Integer} colPos the column position
 * @param {Integer} rowPos the row position
 * @param {Integer} lWord the word longitude
 * @return {boolean} isCollapsed wether if the word is collapsed or not
 */
public boolean isWordCollapsed(Integer orientation, Integer colPos, Integer rowPos, Integer lWord){

    boolean isCollapsed = false;

    try {
        //*Vertical
    if( orientation == 0){
        for(int i = rowPos; i < rowPos + lWord; i++){
            if(soup[i][colPos] != null){
                isCollapsed = true;
            }
        }
    }

    //*Horizontal
    if( orientation == 1){
        for(int  i = colPos; i < colPos + lWord; i++){
            if(soup[rowPos][i] != null){
                isCollapsed = true;
            }
        }
    }

    //*Inverted Vertical
    if( orientation == 2){
        for(int i = rowPos; i > rowPos - lWord; i--){
            if(soup[i][colPos] != null){
                isCollapsed = true;
            }
        }
    }
```

```
//*Inverted Horizontal
if( orientation == 3){
    for(int i = colPos; i > colPos - lWord; i--){
        if(soup[rowPos][i] != null){
            isCollapsed = true;
        }
    }
}

//*Right diagonal
if( orientation == 4){
   for(int i = rowPos, j = colPos; i > ( rowPos - lWord ) && j < ( colPos + lWord ); i--,j++){
       if(soup[i][j] != null){
           isCollapsed = true;
       }
   }
}

//*Left diagonal
if( orientation == 5){
    for(int i = rowPos, j = colPos; i > ( rowPos - lWord ) && j > ( colPos - lWord ); i--,j--){
       if(soup[i][j] != null){
           isCollapsed = true;
       }
   }
}

//*Right diagonal down
if( orientation == 6){
    for(int i = rowPos, j = colPos; i < (lWord + rowPos) && j < (colPos + lWord); i++,j++ ){
        if(soup[i][j] != null){
            isCollapsed = true;
        }
    }
}

    //*Left diagonal down
    if( orientation == 7){
        for(int i = rowPos, j = colPos; i < ( rowPos + lWord ) && j > ( colPos - lWord ); i++,j--){
           if(soup[i][j] != null){
               isCollapsed = true;
           }
       }
    }

    } catch (Exception e) {
        System.err.println("Exception caught");
    }
    return isCollapsed;
}
```

Ilustración 11. Método isWordCollapsed

Método fillSoup

Este método se encarga de llenar los espacios vacíos de la sopa con palabras del alfabeto de manera aleatoria.

```
/**
 * @brief Method that fills the soup with random letters took from the alphabet
 */
public void fillSoup(){
    String alphabet = "abcdefghijklmnñopqrzuv";

      for (int i = 0 ; i<soup.length;i++){
        Random rand = new Random();

        int randomInt = rand.nextInt(alphabet.length());
        for (int j = 0; j<soup[0].length;j++){
            if (soup[i][j] == null){
                soup[i][j] = "" + alphabet.charAt(randomInt);
            }
        }

    }
}
```

Ilustración 12. Método fillSoup

Método showSoup

Este método se encarga de imprimir en pantalla la sopa de letras.

```
/**
 * @brief Method that show the soup
 * @param {String[][]} aSoup The matrix that contains the soup
 */
public void showSoup(String aSoup[][]){
    System.out.println("i\tj 0\t1\t2\t3\t4\t5\t6\t7\t8\t9\t10\t11\t12\t13\t14");
    for (int i = 0 ; i<aSoup.length;i++){
        System.out.print(i);
        for (int j = 0; j<aSoup[0].length;j++){
            if (j!=aSoup[0].length){
                System.out.print("\t" + aSoup[i][j]);
            }

        }
        System.out.println("");
    }
}
```

Ilustración 13. Método showSoup

Cliente

Main

Dentro del main se repite el juego de la sopa de letras mientras el usuario quiera seguir jugando, además de que se manda al servidor la opción del usuario de seguir jugando o no para que el servidor siga a la escucha de este.

```
public static void main(String[] args) {
    System.out.println("**********  SOPA DE LETAS   **********");
    connectGame();
    Integer ans = 1;

    while(ans == 1){
        initGame();
        recvSoup();
        sO.showSoup(soup);
        searchWords();
        recvElapsedTimes();
        System.out.println("Jugar de nuevo? 1 - si / 0 - no");
        ans = scan.nextInt();
        try {
          ByteArrayOutputStream baos = new ByteArrayOutputStream();
            DataOutputStream dos = new DataOutputStream(baos);
            dos.writeInt(ans);
            byte[] b = baos.toByteArray();
            p = new DatagramPacket(b, b.length, dst, pto);
            cl.send(p);
        } catch (Exception e) {
            System.out.println("No se pudo conectar al servidor");
        }
    }
}
```

Ilustración 14. Cliente main

Método connectGame

En este método se le pide al usuario que ponga su username para comenzar a jugar, este nombre se manda al servidor para que este sepa quien se está conectando.

```java
/**
 * @brief Method that connect the client with the server si the client start playinh
 * @param
 * @return
 */
public static void connectGame(){
    String name;

    System.err.println("Introduzca su nombre para empezar a jugar");
    name = scan.nextLine();
    try {
        //*Datagram for client
        cl = new DatagramSocket();
        cl.setReuseAddress(true);
        byte[] b = name.getBytes(); //*Getting bytes of the client name
        dst = InetAddress.getByName(serverAddress);
        //*Creating packet
        DatagramPacket userName = new DatagramPacket(b, b.length, dst, pto);
        cl.send(userName);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 15. Método connectGame

Método initGame

Dentro de este método se manda a llamar la función para que el usuario elija la dificultad, además de que le muestra al usuario un menú con los posibles temas que puede elegir para la sopa de letras, lee la opción del usuario y se manda al servidor que se encargará de crear la sopa de letras de acuerdo con el tema que selecciono el usuario.

De igual forma en este método se reciben las palabras que el usuario pude buscar en la sopa de letras provenientes del servidor, estas se guardan en una lista.

```java
public static void initGame(){
    setDifficulty();
    try {
        System.out.println("Escribe el número del concepto para generar sopa de letas");
        System.out.println("1 - Tecnología");
        System.out.println("2 - Animales");
        System.out.println("3 - Matemáticas");

        //*Sending concept option
        Integer opt = scan.nextInt();
        scan.nextLine();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        dos.writeInt(opt);
        byte[] b = baos.toByteArray();
        p = new DatagramPacket(b, b.length, dst, pto);
        cl.send(p);

        b = new byte[max];
        p = new DatagramPacket(b, max, dst, pto);
        System.out.println("Reciveing words");

        listWords = new ArrayList<String>();
        //*Recieving words
        words = new String[15];
        for(int  i = 0; i < 15; i++){
            cl.receive(p);
            ByteArrayInputStream bais = new ByteArrayInputStream(p.getData());
            ObjectInputStream ois = new ObjectInputStream(bais);
            words[i] = (String) ois.readObject();
            listWords.add(words[i]);
            ois.close();
        }

        showWords(listWords);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 16. Método initGame

Método setDifficulty

Este método simplemente le muestra al usuario un menú con las posibles dificultades que puede elegir, dependiendo de la dificultad que seleccione se manda a llamar la función que manda la dificultad al servidor.

```java
/**
 * @brief Method that sets the game difficulty easy, medium or hard
 * @param
 * @return
 */
public static void setDifficulty(){
    System.out.println("Selecciona la dificultad");
    System.out.println("1 - Fácil: Encontrar 5 palabras en la sopa");
    System.out.println("2 - Medio: Encontrar 10 palabras en la sopa");
    System.out.println("3 - Difícil: Encontrar 15 palabras en la sopa");
    Integer opt = scan.nextInt();
    scan.nextLine();
    switch(opt){
        case 1:
            difficulty = 5;
            sendDif(opt);
            break;
        case 2:
            difficulty = 10;
            sendDif(opt);
            break;
        case 3:
            difficulty = 15;
            sendDif(opt);
            break;
        default:
            difficulty = 5;
            sendDif(opt);
            break;
    }
}
```

Ilustración 17. Método setDifficulty

Método sendDif

Este método se encarga de mandar al servidor la dificultad que el usuario escogió, el envío se realiza debidamente usando sockets de datagrama.

```java
/**
 * @brief Method that sends the difficult to the server
 * @param {Integer} difi the difficult option set by the client
 * @return
 */
public static void sendDif(Integer difi){
    try {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        dos.writeInt(difi);
        byte[] b = baos.toByteArray();
        p = new DatagramPacket(b, b.length, dst, pto);
        cl.send(p);
    } catch (Exception e) {
        System.err.println("No se pudo mandar la dificultad");
    }
}
```

Ilustración 18. Método sendDif

Método showWords

Este método simplemente se encarga de imprimir las palabras que el usuario no ha encontrado en la sopa de letras.

```
/**
 * @brief Method that show the rest of the words that the user can search
 * @param listWords
 */
public static void showWords(List<String> listWords){
    System.out.println(ANSI_PURPLE + "Palabras a encontrar en la sopa" + ANSI_RESET);
    for(String word : listWords)
        System.out.println(ANSI_PURPLE + listWords.indexOf(word)+ " - " + word + ANSI_RESET);

}
```

Ilustración 19. Método showWords

Método recvSoup

Este método se encarga de recibir la sopa de letras creada por el servidor a través de sockets de datagrama. Para esto se creo un arreglo bidimensional donde se almacenará la sopa del servidor. Por cada iteración se manda un carácter almacenado en la sopa en la posición actual del ciclo, esto es eficiente debido a la velocidad de UDP.

```
/**
 * @brief Method that recieves the soup created by the server
 * @param
 * @return
 */
public static void recvSoup(){
    try {
       soup = new String[15][15];
       byte[] b = new byte[max];
        p = new DatagramPacket(b, max, dst, pto);
       for (int i = 0 ; i<soup.length;i++){
            for (int j = 0; j<soup[0].length;j++){
                if (j!=soup[0].length){
                    cl.receive(p);
                    ByteArrayInputStream bais = new ByteArrayInputStream(p.getData());
                    ObjectInputStream ois = new ObjectInputStream(bais);
                    soup[i][j] = (String) ois.readObject();
                }
            }
        }
    } catch (Exception e) {
       System.err.println("No se pudo recibir la sopa del servidor");
    }
}
```

Ilustración 20. Método recvSoup

Método searchWords

Este es el método más complejo pues aquí se encarga de buscar la palabra que el usuario quiere encontrar en la sopa.

Primero se le pide al usuario que elija la palabra a buscar y sus coordenadas de inicio y de fin. Posteriormente, dependiendo de las coordenadas se determina la orientación de la palabra y ya con la orientación determinada podemos ir avanzando posición por posición en la sopa de letras para ver si coincide con el carácter de la palabra en esa posición actual del ciclo. Si es encontrada, se le avisa al usuario y su puntaje aumenta en uno, mostrándole las palabras restantes por buscar de acuerdo con la dificultad que eligió al inicio del juego.

En caso de encontrar la palabra se le vuelve a mostrar las palabras y la sopa de letras.

```
/**
 * @brief Method that search for the words the user wanna search
 * @param
 * @return
 */
public static void searchWords(){
    System.out.println("");
    Integer correct = 0;
    Scanner coord = new Scanner(System.in);
    boolean found = false;
    long initTime;
    long endTime;

    List<String> wordsFound = new ArrayList<String>();
    //*Checkind the time the user began to solve the soup
    initTime = System.nanoTime();

    while(correct != difficulty){
        System.out.println("Ingrese el numero de la palabra a buscar");
        Integer indexWord = scan.nextInt();

        if(indexWord < listWords.size()){
            //String wordToSearch = words[indexWord];
            String wordToSearch = listWords.get(indexWord);

            //*Verifying if word has been already found
            if(wordsFound.contains(wordToSearch)){
                System.err.println("Palabra ya encontrada, trate con otra");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
                }
            }else{
                System.out.println("Ingrese la coordenada (i,j) de inicio y fin de la palabra que quieres buscar en la sopa");
                System.out.println(ANSI_BLUE + "Coordenada de inicio" + ANSI_RESET);
                System.out.print("i: ");
                Integer i1 = coord.nextInt();
                coord.nextLine();
                System.out.println("");
                System.out.print("j: ");
                Integer j1 = coord.nextInt();
                coord.nextLine();
                System.out.println(ANSI_BLUE + "Coordenada de fin" + ANSI_RESET);
```

```
System.out.print("i: ");
Integer i2 = coord.nextInt();
coord.nextLine();
System.out.println("");
System.out.print("j: ");
Integer j2 = coord.nextInt();
coord.nextLine();
System.out.println("");

//*Searching if word is horizontal
if(il == i2 && j2 > j1){
    for(Integer l = 0, m = j1; m <= j2 && l < wordToSearch.length(); l++, m++){
          System.out.println("Sopa " + soup[il][m] + " palabra " + wordToSearch.charAt(l) );
        if(soup[il][m].equals(""+wordToSearch.charAt(l))){
            found = true;
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
        } else {
            System.err.println("Palabra no encontrada, trate con otra");
            found = false;
            break;
        }
    }
    if(found)
        correct++;
}
//*Searching if word is inverted horizontal
if(il == i2 && j1 > j2){
    for(Integer l = 0, m = j1; m >= j2 && l < wordToSearch.length(); l++, m--){
          System.out.println("Sopa " + soup[il][m] + " palabra " + wordToSearch.charAt(l) );
        if(soup[il][m].equals(""+wordToSearch.charAt(l))){
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
            found = true;
        } else {
            System.err.println("Palabra no encontrada, pruebe de nuevo");
            found = false;
            break;
        }
    }
    if(found)
        correct++;
```

```
//*Searching if word is vertical
if(j1 == j2 && i2 > i1){
    for(Integer l = 0, m = i1; m <= i2 && l < wordToSearch.length(); l++, m++){
          System.out.println("Sopa " + soup[m][j1+ " palabra " + wordToSearch.charAt(l) );

        if(soup[m][j1].equals(""+wordToSearch.charAt(l))){
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
            found = true;
        } else {
            System.err.println("Palabra no encontrada, pruebe de nuevo");
            found = false;
            break;
        }
    }

    if(found)
        correct++;
}

//*Searching if word is inverted vertical
if(j1 == j2 && i1 > i2){
    for(Integer l = 0, m = i1; m >= i2 && l < wordToSearch.length(); l++, m--){
          System.out.println("Sopa " + soup[i1][m] + " palabra " + wordToSearch.charAt(l) );
        if(soup[m][j1].equals(""+wordToSearch.charAt(l))){
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
            found = true;
        } else {
            System.err.println("Palabra no encontrada, pruebe de nuevo");
            found = false;
            break;
        }
    }
    if(found)
        correct++;
}
```

```
//*Searching if word is right diagonal
if(j2 > j1 && i1 > i2){
    for(Integer l = 0, m = i1, n = j1; m >= i2 && n <= j2 && l < wordToSearch.length(); l++, m--, n++){
          System.out.println("Sopa " + soup[m][n] + " palabra " + wordToSearch.charAt(l) );
        if(soup[m][n].equals(""+wordToSearch.charAt(l))){
            found = true;
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
        } else {
            System.err.println("Palabra no encontrada, pruebe de nuevo");
            found = false;
            break;
        }
    }
    if(found)
        correct++;
}

//*Searching if word is left diagonal downS
if(i2 > i1 && j1 > j2){
    for(Integer l = 0, m = i1, n = j1; m <= i2 && n >= j2 && l < wordToSearch.length(); l++, m++, n--){
          System.out.println("Sopa " + soup[m][n] + " palabra " + wordToSearch.charAt(l) );
        if(soup[m][n].equals(""+wordToSearch.charAt(l))){
            found = true;
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
        } else {
            System.err.println("Palabra no encontrada, pruebe de nuevo");
            found = false;
            break;
        }
    }
    if(found)
        correct++;
}

//*Searching if word is left diagonal
if(i1 > i2 && j1 > j2){
    for(Integer l = 0, m = i1, n = j1; m >= i2 && n >= j2 && l < wordToSearch.length(); l++, m--, n--){
          System.out.println("Sopa " + soup[m][n] + " palabra " + wordToSearch.charAt(l) );
        if(soup[m][n].equals(""+wordToSearch.charAt(l))){
            found = true;
            wordsFound.add(wordToSearch);
            listWords.remove(wordToSearch);
```

Ilustración 21. Método searchWords

Método sendSolveTime

Este método manda al servidor el tiempo total que le tomo al usuario resolver la sopa de letras en minutos.

Ilustración 22. Método sendSolveTime

Método recvElapsedTimes

Este método recibe desde el servidor los tiempos que han obtenido todos los jugares que han jugado a la sopa de letras, se le muestra el tiempo, a la dificultad y con el tema que eligieron para jugar.

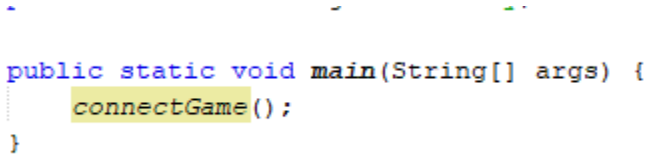
```
/**
 * Brief Method that recieves the other user times
 * @param
 * @return
 */
public static void recvElapsedTimes(){
    try {
        byte[] b = new byte[max];
        p = new DatagramPacket(b, max, dst, ptn);
          System.out.println("Recieving elapsed Time");

        //*Recieving times
        List<String> usersTimes = new ArrayList<String>();
        Integer listSize;
        cl.receive(p);
        ByteArrayInputStream bais = new ByteArrayInputStream(p.getData());
        ObjectInputStream ois = new ObjectInputStream(bais);
        listSize = ois.readInt();
        ois.close();

        for(Integer  i = 0; i < listSize; i++){
            cl.receive(p);
            bais = new ByteArrayInputStream(p.getData());
            ois = new ObjectInputStream(bais);
            usersTimes.add((String) ois.readObject());
            ois.close();
        }

        System.out.println(ANSI_PURPLE + "Tiempo de otros usuarios" + ANSI_RESET);
        for(String time : usersTimes)
            System.out.println(ANSI_PURPLE + time + ANSI_RESET);
          for(int i = 0; i< usersTimes.size();i++)
              System.out.println(ANSI_PURPLE + usersTimes.get(i) + ANSI_RESET);

    } catch (Exception e) {
        System.err.println("No se pudieron recibir los tiempos");
    }
}
```

Ilustración 23. Método recvElapsedTimes

Servidor

Main

En el main solo mandamos a llamar la función connectGame



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

Ilustración 24. Main del servidor

Método connectGame

En este método se hace la aceptación del cliente recibiendo el nombre de usuario y un ciclo infinito para recibir datagramas, además de un ciclo que dependiendo si el jugador quiere seguir jugando cierra o no el socket.

```
/**
 * @brief Method that makes the connection with the client and wait for connections
 * @param
 * @return
 */
public static void connectGame() {
    try {
        s = new DatagramSocket(pto);
        s.setReuseAddress(true);
        System.out.println("Server Iniciado");
        //*Forever to recieve datagrams
        for (;;) {
            DatagramPacket p = new DatagramPacket(new byte[max], max);
            //*Recieving client username
            s.receive(p);
            userName = new String(p.getData(), 0, p.getLength());
            System.err.println(userName + " Conectado");
            //*Creating client connection
            s.connect(p.getSocketAddress());
            Integer opt = 1;

            while(opt == 1){
                setDifficulty();
                setConcept();
                createSoup(words);
                recvElapsedTime();
                sendUserTimes();
                p = new DatagramPacket(new byte[max], max);
                s.receive(p);
                DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
                opt = dis.readInt();
            }

            s.close();

        }
    } catch (Exception e) {

    }
}
```

Ilustración 25. Método connectGame

Método setDifficulty

Este método se encarga de establecer la dificultar del juego de acuerdo a la que escogió el usuario al inicio.

```
/**
 * @brief Method that recieves the difficult set by the client
 * @param
 * @return
 */
public static void setDifficulty(){
    try {
        DatagramPacket p = new DatagramPacket(new byte[max], max);
        s.receive(p);
        DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
        int opt = dis.readInt();
          switch(opt){
        case 1:
            difficulty = "facil";

            break;
        case 2:
            difficulty = "medio";

            break;
        case 3:
            difficulty = "dificil";

            break;
        default:
            difficulty = "facil";
            break;
    }
    } catch (Exception e) {
        System.err.println("No se pudo recibir la dificultad");
    }

}
```

Ilustración 26. Método setDifficulty

Método setConcept

Aquí establecemos de que archivo se agarrarán las palabras para la sopa de letras de acuerdo con el tema escogió por el usuario. Se recibe la opción por medio de un paquete y con un swtich determinamos el concepto del juego.

```java
/**
 * @brief Method that set the concept choosen by the client
 * @param
 * @return
 */
public static void setConcept() {
    try {
        DatagramPacket p = new DatagramPacket(new byte[max], max);
        s.receive(p);
        DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
        int opt = dis.readInt();
        dst = p.getAddress();
        switch (opt) {
            case 1:
                System.out.println("Tecnologia seleccionado");
                getConceptWords("tec.txt");
                topic = "tecnologia";
                  createSoup(words);

                break;
            case 2:
                System.out.println("Animales seleccionado");
                getConceptWords("ani.txt");
                topic = "animales";
                break;
            case 3:
                System.out.println("Matematicas seleccionado");
                getConceptWords("mate.txt");
                topic = "matematicas";
                break;
            default:
                break;
        }
    } catch (Exception e) {
    }
}
```

Ilustración 27. Método setConcept

Método getConceptWords

En este método obtenemos las palabras del archivo que corresponde al concepto seleccionado por el usuario y así como se van leyendo las palabras se van enviando al cliente por medio de paquetes hasta que ya no haya más palabras.

```
public static void getConceptWords(String filename) throws IOException {
    Integer i = 0;

    try{
          scan = new Scanner(new File(path+filename));
          words = new String[15];
          while(i < words.length){
              //*Reading word from file
              String line = scan.nextLine();
  //            System.out.println(i + " " + line);
              //*Saving words
              words[i] = line;
                        //*Sending words to client
              baos = new ByteArrayOutputStream();
              oos = new ObjectOutputStream(baos);
              oos.writeObject(words[i]);
              oos.flush();
              byte[] buff = baos.toByteArray();
              baos.flush();
              DatagramPacket packet = new DatagramPacket(buff, buff.length);
              s.send(packet);

  //            System.out.println(i + " " + words[i]);
              i++;
          }

  //            System.out.println("Sending words");

          }catch(Exception e){
              System.err.println("No se pudieron enviar palabras als ervidor");
          }
}
```

Ilustración 28. Método getConceptWords

Método createSoup

Este método se encarga de crear, setar y llenar la sopa con ayuda de los métodos ya vistos de la sopa de letras, de igual forma en este método se hace el envió de la sopa de letras.

```java
/**
 * @brief Method that creates the soup and send it to the user
 * @param {String[]} words The words that are gonna be included in the soup
 * @return
 */
public static void createSoup(String[] words){
    Soup so = new Soup(15, 15, words);
    so.setSoup();
    so.fillSoup();

    soupToSend = new String[15][15];
    soupToSend = so.getSoup();
    //*Sending the created soup to the client
    try {
        for (int i = 0 ; i<soupToSend.length;i++){

            for (int j = 0; j<soupToSend[0].length;j++){
                if (j!=soupToSend[0].length){
                    baos = new ByteArrayOutputStream();
                    oos = new ObjectOutputStream(baos);
                    oos.writeObject(soupToSend[i][j]);
                    oos.flush();
                    byte[] buff = baos.toByteArray();
                    baos.flush();
                    DatagramPacket packet = new DatagramPacket(buff, buff.length);
                    s.send(packet);
                }

            }

        }
    } catch (Exception e) {
    }

}
```

Ilustración 29. Método createSoup

Método recvElapsedTime

Este método recibe el tiempo que tardo el usuario para guardarlo al final de un archivo que contiene todos los tiempos históricos de los jugadores que han jugado a la sopa de letras, además se guarda el nombre del usuario que hizo ese tiempo, en la dificultad que lo hizo y en que concepto jugó.

```
 * @brief Method that recieves the user elapsed time
 * @param
 * @return
 */
public static void recvElapsedTime(){

    records = new File("C:\\Users\\PC\\Desktop\\6to Semestre ESCOM\\Aplicaciones para comunicaciones en red\\Practic

    try {

        DatagramPacket p = new DatagramPacket(new byte[max], max);
        s.receive(p);
        DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
        double elapsedTime = dis.readDouble();
        System.err.println("Jugador " + userName + " tardo " + elapsedTime);

        //Writing record in
        FileWriter fw = new FileWriter(records, true);
        BufferedWriter bw = new BufferedWriter(fw);

        bw.write(userName + ": " + String.valueOf(elapsedTime) + " minutos - " + topic + " - " + difficulty);
        bw.newLine();

        bw.close();
        fw.close();

    } catch (Exception e) {
        System.err.println("No se pudo recibir el tiempo del usuario");
    }
}
```

Ilustración 30. Método recvElapsedTime

Método sendUserTimes

Este método se encarga de leer los registros de tiempos de todos los jugadores para mandarlos a través de paquetes al cliente para que este pueda ver los tiempos de los demás jugadores y compararlos con el suyo.

Ilustración 31. Método sendUserTimes

Pruebas de funcionamiento

Al correr el servidor este indica que ha sido inicializado y el cliente le pregunta al usuario su nombre como se ve en la siguiente ilustración.

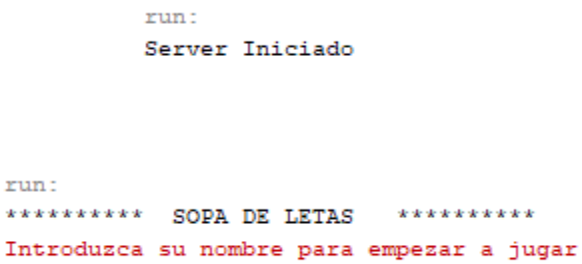


Ilustración 32. Corriendo el servidor y el cliente.

Al introducir el nombre el cliente, el servidor lo recibe y se le pregunta al usuario la dificultad que va a seleccionar como se ve en la siguientes ilustraciónes.

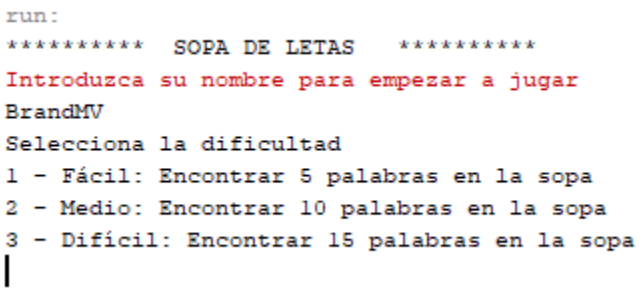


Ilustración 33. Usuario introduciendo su nombre

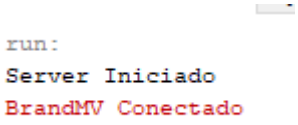


Ilustración 34. Servidor indicando quien se ha conectado.

Al escoger el usuario la dificultad se le pregunta al usuario el concepto que quiere para la sopa de letras, al elegirlo se muestra la sopa de letras junto con las palabras que puede buscar, el servidor recibe el concepto como se ve en las siguientes ilustraciones.

```
Selecciona la dificultad
1 - Fácil: Encontrar 5 palabras en la sopa
2 - Medio: Encontrar 10 palabras en la sopa
3 - Difícil: Encontrar 15 palabras en la sopa
1
Escribe el nùmero del concepto para generar sopa de letas
1 - Tecnología
2 - Animales
3 - Matemáticas
2
Reciveing words
Palabras a encontrar en la sopa
0 - antilope
1 - avestruz
2 - boa
3 - bongo
4 - bufalo
5 - cebra
6 - camaleon
7 - gato
8 - perro
9 - cerdo
10 - chimpance
11 - rana
12 - cocodrilo
13 - cucaracha
14 - elefante
i       j 0     1       2       3       4       5       6       7       8       9       10      11      12      13      14
0       r       r       a       h       c       a       r       a       c       u       c       r       r       r       r
1       e       c       n       a       p       m       i       h       c       j       c       j       j       j       j
2       j       j       j       o       g       n       o       b       j       o       j       j       j       j       j
3       u       u       u       u       u       u       u       u       c       o       u       u       a       u       u
4       q       b       u       f       a       l       o       o       t       q       q       n       q       q       z
5       n       n       n       n       n       n       d       a       n       n       t       n       n       u       n
6       a       r       b       e       c       r       g       u       u       i       u       u       r       u       u
7       e       e       e       e       i       e       e       e       l       e       e       t       e       e       e
8       p       p       p       l       p       p       p       o       p       p       s       p       p       p       p
9       o       o       o       o       b       o       p       o       o       e       o       o       o       p       o
10      b       b       b       o       b       e       b       b       v       b       b       b       b       e       b
11      c       c       a       c       c       c       c       a       c       c       c       c       c       r       c
12      n       o       e       l       a       m       a       c       g       g       g       g       g       r       g
13      e       t       n       a       f       e       l       e       a       n       a       r       l       o       l
14      i       i       o       d       r       e       c       i       i       i       i       i       i       i       i

Ingrese el numero de la palabra a buscar
```

Ilustración 35. Sopa de letras mostrada

```
run:
Server Iniciado
BrandMV Conectado
Dificultad: facil
Animales seleccionado
```

Ilustración 36. Información del servidor.

A partir de esto el cliente ya puede ingresar la palabra que quiere buscar y se le pedirán las coordenadas de inicio y fin, si la palabra fue encontrada se quitará de la lista a buscar y se el aumenta la puntuación como se puede ver en la siguiente ilustración.

```
Ingrese el numero de la palabra a buscar
6
Ingrese la coordenada (i,j) de inicio y fin de la palabra que quieres buscar en la sopa
Coordenada de inicio
i: 12

j: 7
Coordenada de fin
i: 12

j: 0

Palabras encontradas 1 restantes 4
Palabras a encontrar en la sopa
0 - antilope
1 - avestruz
2 - boa
3 - bongo
4 - bufalo
5 - cebra
6 - gato
7 - perro
8 - cerdo
9 - chimpance
10 - rana
11 - cocodrilo
12 - cucaracha
13 - elefante
```

Ilustración 37. Encontrando una palabra.

En caso de que el usuario quiera buscar una palabra que no esta en la lista se le mostrará un error.

```
Ingrese el numero de la palabra a buscar
16
Palabra no valida, pruebe con otra
Ingrese el numero de la palabra a buscar
```

Ilustración 38. Mensaje de error

De igual forma si las coordenadas que ingresa no corresponden a la palabra que quiere buscar se le mandará un error.

```
Ingrese el numero de la palabra a buscar
3
Ingrese la coordenada (i,j) de inicio y fin de la palabra que quieres buscar en la sopa
Coordenada de inicio
i: 1

j: 4
Coordenada de fin
i: 2

j: 1

Palabra no encontrada, pruebe de nuevo
```

Ilustración 39. Mensaje de error con coordenadas no válidas.

Cuando el usuario encuentra todas las palabras que debe buscar de acuerdo con la dificultad elegida se le muestra el tiempo total que hizo y el tiempo de los demás jugadores y se le pregunta si quiere jugar de nuevo, de igual forma el servidor recibe el tiempo que hizo este usuario como se ve en las siguientes ilustraciones.

```
Tu tiempo: 9.904203265 minutos
Tiempo de otros usuarios
pedro: 5 minutos
ruben: 10 minutos
brandon: 30 minutos
roberto: 1000 minutos
Brand: 3.5553521216666666 minutos
pedrito: 2.850987981666667 minutos
Josue: 3.2978069199999998 minutos - matematicas - facil
BrandMV: 9.904203265 minutos - animales - facil
Jugar de nuevo? 1 - si / 0 - no
```

Ilustración 40. Finalizando el juego

```
run:
Server Iniciado
BrandMV Conectado
Dificultad: facil
Animales seleccionado
Jugador BrandMV tardo 9.904203265
```

Ilustración 41. Tiempo capturado en servidor.

Si el jugador selecciona que no quiere jugar más se cierra el socket.

```
run:
Server Iniciado
BrandMV Conectado
Dificultad: facil
Animales seleccionado
Jugador BrandMV tardo 9.904203265
BUILD SUCCESSFUL (total time: 11 minutes 6 seconds)


Jugar de nuevo? 1 - si / 0 - no
0
BUILD SUCCESSFUL (total time: 11 minutes 0 seconds)
```

Ilustración 42. Cerrando conexión.

Conclusiones

Saber sobre sockets de datagrama es muy importante pues son usados en aplicaciones done se requiere velocidad al enviar paquetes, un ejemplo es el juego desarrollado en esta práctica, pues queremos que el usuario reciba la mayor experiencia y que el juego sea rápido en el envío y recibimiento de datos.

Gracias a esta práctica pude comprender de una mejora manera el uso de los sockets de datagrama usando, en este caso, lenguaje Java para su aplicación en un juego de sopa de letras donde el envió y recibo de datos fue constante. Sin dudas fue una buena práctica para poner en práctica lo visto en clase relacionado a los sockets de datagrama y ver las diferencias con los sockets de flujo vistos en la práctica anterior.

Durante el desarrollo de esta práctica presente varios problemas, sobre todo al momento de mandar la sopa de letras, pues había momentos en los que nunca se enviaba, esto lo resolví mandando la sopa de letras ya creada para después recibirla en el cliente. De igual forma tuve conflicto en la lógica de buscar las palabras en la sopa de letras a partir de las coordenadas de inicio y fin, esto lo resolví de buena forma identificando la orientación de la palabra y a partir de unas condiciones con las coordenadas poder buscar la palabra de una manera eficiente en la sopa de letras.

Esta práctica represento un reto y fue divertido de solucionar, además de que me llevo un conocimiento reforzado sobre los sockets de datagrama.

Bibliografía

1) Froufe, A. (2020). “Sockets de datagrama”. Obtenido de: https://mate.uprh.edu/~jse/cursos/4097/notas/java/javaEspanol/JavaTut/Cap9/socket.html