TECNOLÓGICO NACIONAL DE MÉXICO

campus LEÓN

TOPICOS AVANZADOS DE PROGRAMACION

REPORTE: Laberinto con back tracking (Laberinto Maze)

DESARROLLADOR:

- Conriquez Cabrera Marco Antonio

DOCENTE:

ING. LEVY ROJAS CARLOS RAFAEL

CARRERA:

INGENIERÍA EN SISTEMAS COMPUTACIONALES

Fecha: 20/04/2023 **León, Guanajuato**

Índice

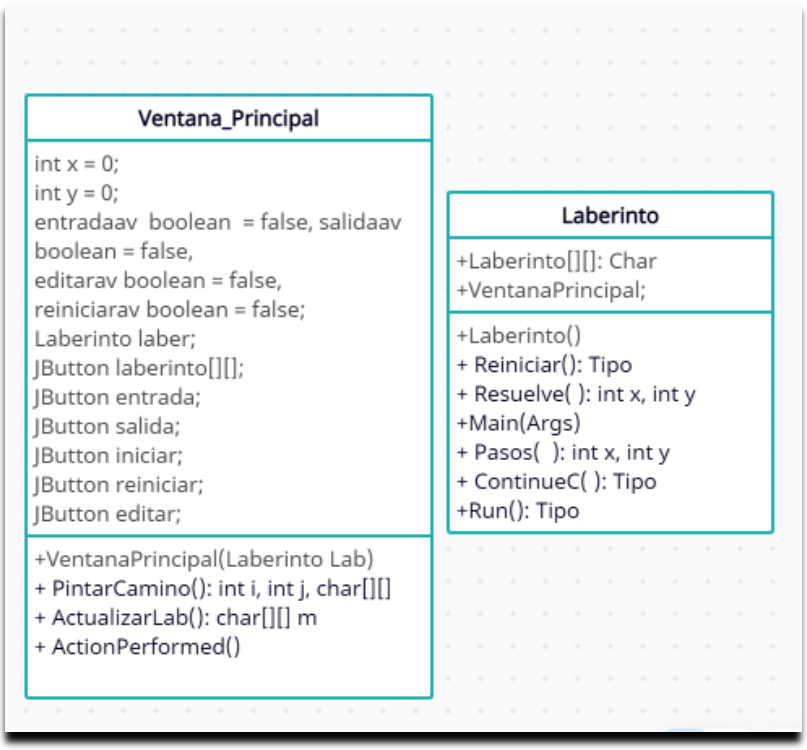
DIAGRAMAS UML **3**

Desarrollo y Códigos fuentes **8**

Conclusion.....9

Bibliografía **10**

DIAGRAMAS UML DE LA APLICACIÓN



DESARROLLO DE LA APLICACIÓN:

Para la clase Laberinto contiene distintos métodos, los cuales son más que nada para el funcionamiento del Laberinto.

Primero se utilizó el Extends Thread que sirve para la creación de una clase hija a otra. Y la clase Thread quiere decir que se va a extender para crear una nueva clase para que herede sus métodos y campos.

Después se utilizó como variable el Public char[][] Laberinto

Para así poder asignarle una base de el laberinto que estaremos desarrollando como preterminado.

```java
//DESARROLLADO POR MARCO ANTONIO CONRIQUEZ CABRERA

public class Laberinto extends Thread {

        public VentanaPrincipal VentanaP ;
        public char[][] laberinto={
        {'1','1','1','1','1','1','1','1','1','1'},
        {'1','0','1','0','0','0','0','0','0','1'},
        {'1','0','0','0','1','0','0','1','0','1'},
        {'1','1','1','1','0','0','1','1','1','1'},
        {'1','0','0','0','0','1','0','0','0','1'},
        {'1','0','1','1','1','1','0','1','0','1'},
        {'1','0','0','0','0','1','0','1','0','1'},
        {'1','1','0','1','1','1','0','1','0','1'},
        {'1','0','0','0','0','0','0','1','0','1'},
        {'1','1','1','1','1','1','1','1','1','1'},
                };
```

Después la parte heredada se agregaron todas las características de diseño, pero solo para el JFrame, El diseño de texto, tamaño y ubicación de él.

También se hace llamado a la otra clase que es “ VentanaP = Ventana Principal”.

```java
Laberinto()
{
 JLabel Instrucciones1,Instrucciones2,Titulo, Desarrollador;
 VentanaP = new VentanaPrincipal(this);
 VentanaP .actualizarLab(laberinto);
 //DISEÑO,POSICION Y TAMAÑO DEL JFRAME
 VentanaP .setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Para cerrar la aplicacion directamente
 Dimension ss = Toolkit.getDefaultToolkit ().getScreenSize ();
 Dimension frameSize = new Dimension ( 900, 690 );
 VentanaP .setLayout(null);
 VentanaP .setVisible(true);
 VentanaP . setResizable(false);
 VentanaP .setBounds ( ss.width / 2 - frameSize.width / 2, ss.height / 2 - frameSize.height / 2,
 frameSize.width, frameSize.height );
 //INSTRUCCIONES CON LABEL
 Titulo = new JLabel("LABERINTO RECURSIVO");
 Titulo.setBounds(525,10,400,60);
 Titulo.setForeground(Color.darkGray);
 Titulo.setFont(new Font("Serif", Font.BOLD, 20));
 VentanaP .add(Titulo);
 Instrucciones1 = new JLabel("1-Por favor seleccione entrada y salida y ubiquelos en el laberinto ");
 Instrucciones1.setBounds(100,560,500,40);
 VentanaP .add(Instrucciones1);
 Instrucciones2 = new JLabel("2- Si quiere editar, debe ir selecionando el boton editar y click haga cli
 Instrucciones2.setBounds(100,580,600,40);
 VentanaP .add(Instrucciones2);
}
```

Método Reiniciar

```
public void reiniciar() {
        for(int i = 0; i < 10; i++){
         for(int j = 0; j < 10; j++){
            if(laberinto[i][j] != '1'){ // Los for recorren las filas y columnas
               laberinto[i][j] = '0';
             }
           }
          }
}
```

-En el método reiniciar simplemente es para recorrer y elaborar el laberinto como el preterminado.

En la clase "Laberinto" es tipo Main, así que aquí se depura parte del programa.

```
public static void main(String[] args)
{
      Laberinto m = new Laberinto();
    }
```

Ahora con los siguientes métodos se hace implementación del Algoritmo:

Resuelve:

```
public void resuelve(int x, int y)
{
     if(pasos(x,y))     //Permite introducir las coordenadas (x,y)
    {
     laberinto[x][y] = 'S';  //Introduce en las coordenadas x,y la entrada
    }
    else
    {
       JOptionPane.showMessageDialog(null, "No hay solucion en el laberinto");
    }
}
```

Pasos:

```
private boolean pasos(int x, int y)
{
    try{
        Thread.sleep(50);  //Para que el programa corra
    } catch (Exception y){

    }
```

Primero se utilizo un Try catch, el cual incluira el Thread.sleep que permitira cambiar la velocidad de la depuracion de el programa.

Despues se agregaron dos condiciones:

if (laberinto[x][y]=='S'){ // si hemos llegado a X quiere decir que hemos encontrado la solución

JOptionPane.showMessageDialog(null, "Se a encontrado la salida del laberinto");

return true; // Termina el algoritmo

}

if (laberinto[x][y]=='1'||laberinto[x][y]=='*') { // si llegamos a una pared o al mismo punto,

return false; // entonces el laberinto no puede resolverse y termina.

}

Después de las condiciones se utilizó el método Backtracking y en cada uno de los pasos que se realizaban se iba utilizando un método que es "Pintarcamino" que lo que hace es ir actualizando el camino y pintándolo, dependiendo del backtracking.

```
// si no se cumple ninguna de estas dos situaciones, quiere decir que nos encontramos en
// caso intermedio, por lo tanto, que empezamos a recorrer o todavía no hemos llegado a
laberinto[x][y]='*'; // marcamos la posición como visitada (si es la primera, en las coo

boolean result; // se coloca S de START)
VentanaP.pintarcamino(x,y,laberinto);
result=pasos(x-1, y); // intentamos ir hacia ARRIBA.
VentanaP.pintarcamino(x,y,laberinto);
if (result)
return true; // si el resultado es el final, entonces el algoritmo termina
VentanaP.pintarcamino(x,y,laberinto);
result=pasos(x, y+1); // intentamos ir hacia la DERECHA.
VentanaP.pintarcamino(x,y,laberinto);
if (result)
return true; // si el resultado es el final, entonces el algoritmo termina
VentanaP.pintarcamino(x,y,laberinto);
result=pasos(x, y-1); // intentamos ir hacia la IZQUIERDA.
VentanaP.pintarcamino(x,y,laberinto);
if (result)
return true; // si el resultado es el final, entonces el algoritmo termina
VentanaP.pintarcamino(x,y,laberinto);
result=pasos(x+1, y); // intentamos ir hacia ABAJO.
VentanaP.pintarcamino(x,y,laberinto);
if (result)
return true; // si el resultado es el final, entonces el algoritmo termina
VentanaP.pintarcamino(x,y,laberinto);
```

```
}

public synchronized void continueC(){
    notifyAll();
}
public synchronized void run(){  //Por la Thread, ejecutar principalmente
   while(true){
       resuelve(VentanaP.x,VentanaP.y);
       try{
           wait();
       } catch (Exception yy){

       }
```

Por último el método Run que simplemente se utiliza el try catch y se llama el método Resuelve, así para determinar que esta bien el programa y no marque ningún error durante el uso del programa.

CLASE VENTANA PRINCIPAL

Para la clase ventana principal solo es para el apartado de diseño de el laberinto, tanto su diseño base como el diseño al ir actualizando el camino.

```
public VentanaPrincipal(Laberinto laber)
{

    this.laber = laber;
    laberinto = new JButton[laber.laberinto.length][laber.laberinto.length];
    //CREE LOS BOTONES EDITAR,ENTRADA Y REINICIAR
    editar = new JButton("Editar");
    editar.setBounds(525,100,90,30);
    editar.addActionListener(this);
    editar.setBackground(Color.gray);
    this.add(editar);

    entrada = new JButton("Entrada");
    entrada.setBounds(50,530,90,30);
    entrada.addActionListener(this);
    entrada.setBackground(Color.green);
    this.add(entrada);

    salida = new JButton("Salida");
    salida.setBounds(150,530,90,30);
    salida.addActionListener(this);
    salida.setBackground(Color.red);
    this.add(salida);
```

Primero en la clase Ventana principal como parámetros se asigno “Laberinto Laber”.

Después dentro del muestra el diseño de cada uno de los componentes dentro del JFrame que viene siendo los botones. Se encuentra el botón Editar, Entrada, Salida, Reiniciar, Iniciar.

```
}
public void actualizarLab(char[][] m){   //Pintar las paredes e inicios del lab
    for(int i = 0; i < m[0].length; i++){
        for(int j = 0; j<m.length;j++){
          if(m[i][j] == '1')
          {
              laberinto[i][j].setText("1");
              laberinto[i][j].setForeground(Color.white);
              laberinto[i][j].setBackground(Color.darkGray);
          }
          else if (m[i][j] == '0')
          {
              laberinto[i][j].setText("0");
              laberinto[i][j].setBackground(Color.white);
          }
          else if (m[i][j] == 'S')
          {
              laberinto[i][j].setBackground(Color.green);

          }
          else if (m[i][j] == 'X')
          {
              laberinto[i][j].setBackground(Color.yellow);

          }
          repaint();
        }
```

Para el método actualizarLab se usa como parámetro el char[][] m para poder hacer cambios de las columnas y filas de el laberinto.

Después con los ciclos y condiciones se muestra el color de fondo de cada botón, en este caso si hay pared, si es el inicio de laberinto, la salida y cuando va avanzando.

```
public void pintarcamino(int i, int j, char[][] m){
    if(!(i == x && j == y)){
      if(m[i][j] == '*')
          laberinto[i][j].setBackground(Color.yellow); //Marca de amarillo la posicion visitada
      else if (m[i][j] == '+')
      laberinto[i][j].setBackground(Color.blue); //Azul cuando el camino fue recorrido pero regreso
    }
    repaint();
}
```

En el método pintar camino solo se agregan las condiciones de el color cuando avanza el laberinto, si avanza se estará asignando como dato un * pero si choca y regresa cambiará de color a azul que será asignado como +.

Por ultimo se van llamando cada uno de los datos y se van agregando para la función de los botones, también las condiciones de diseño a la hora de editar el laberinto.

```
if(e.getSource() == entrada)
    entradaav = true;
if(e.getSource()== editar)
    editarav = true;
if(e.getSource()== salida)
    salidaav = true;
if(e.getSource()== reiniciar)
    reiniciarav = true;

for(int i = 1; i<laberinto.length - 1; i++){
   for(int j = 1; j< laberinto.length - 1; j++){
      if(e.getSource() == laberinto[i][j]){
         if(salidaav){
            laber.laberinto[i][j] = 'S';
            laberinto[i][j].setBackground(Color.red);
            laberinto[i][j].setText("S");  //S de salida
            salidaav = false;
         } else if(entradaav){
            laber.laberinto[i][j] = 'E';
             laberinto[i][j].setBackground(Color.green);
             laberinto[i][j].setText("E"); //E de entrada
             x = i;
             y = j;
             entradaav = false;
         }
         if(editarav){
            if(laber.laberinto[i][j] != '1'){
               laberinto[i][j].setBackground(Color.darkGray);
               laberinto[i][j].setText("1");
               laberinto[i][j].setForeground(Color.white);
               laber.laberinto[i][j] = '1';
```

Conclusión

El desarrollo de esta aplicación fue muy importante para retroalimentar los conocimientos de estructura de datos tanto como en tópicos. Se aprendió el uso de recursividad con ciertos parámetros o variables que permitieron saber el backtracking.

Fuentes bibliográficas

Diaz, A. I. (2014). Algoritmo de Backtracking Recursivo y no Recursivo para la Resolución de un Laberinto y su Aplicación en SDL. *Grenoble-inp*. https://www.academia.edu/4607412/Algoritmo_de_Backtracking_Recursivo_y_no_Recursivo_para_la_Resoluci%C3%B3n_de_un_Laberinto_y_su_Aplicaci%C3%B3n_en_SDL

DiscoDurodeRoer. (2018, April 17). *Ejercicios Java - Recursividad #13 - ¡Caminos posibles de un laberinto con backtracking!* [Video]. YouTube. https://www.youtube.com/watch?v=v0ghwiepPwk

CRISTHIAN RODRIGUEZ. (2018, August 28). *Explicacion del programa del laberinto con Backtracking - java* [Video]. YouTube. https://www.youtube.com/watch?v=oSm6uMxn3h