Universidad de San Carlos de Guatemala

Centro Universitario de Occidente

División de Ciencias de la Ingeniería

Área Profesional

Introducción a la Programación y Computación 1

Sección “A”

Ing. José Moisés Granados Guevara

“MANUAL TECNICO PROYECTO 2”

Melvin Eduardo Ordoñez Sapón RA | 202230552

Quetzaltenango, mayo del 2024

“Id y enseñad a todos”

ACERCA DEL PROGRAMA:

IDE utilizado: Apache NetBeans IDE 20

JDK utilizado: Versión 21.0.2.0

Versión Del Programa: 1.0

Programado en Sistema Operativo: Windows 11 Pro

Terminal Utilizada: Terminal de Microsoft Store

Lenguaje de Programación Utilizado: Java

Compatibilidad: Windows, Linux, macOS

Plugin: maven-assembly-plugin

METODOS IMPORTANTES

Este es uno de los principales ya que inicializa las variables de la clase ControladorPartida en base al laberinto que se le es enviado desde su constructor

```
   //CONSTRUCT0R
public ControladorPartida(JPanel panelTablero, VentanaJuego   ventanaJuego, Laberinto laberintoSeleccionado) {
this.panelTablero = panelTablero;
this.ventanaJuego = ventanaJuego;
this.laberintoSeleccionado = laberintoSeleccionado;
wuacamoleIniciados = 0;
siGano = false;
}
```

SEGUIDO DE ESO VIENE OTRO METODO IMPORTANTE QUE ESTA EN LA MISMA CLASE. LLAMADA INICIACOMPONENTES, CON ESTE METODO SE CREAN EL LABERINTO CON TODA LA INFORMACION QUE RECIBIO EL CONSTRUCTOR Y SEGUIDO DE ESO SE MUESTRAN LOS CAMBIOS REALIZADOS AL FRONTEND

```
public void iniciarComponentes() throws ListaException {
    String tablero = laberintoSeleccionado.getConfiguracionLaberinto();

    //Se aumenta las veces que se ha jugado ese laberinto
    laberintoSeleccionado.aumentarVecesJugado();
```

```
        //Para obtener  un nombre que identifique a esta partida
        VentanaPedirNombrePartida pedirNombre = new
VentanaPedirNombrePartida(ventanaJuego, true, this);
        pedirNombre.setVisible(true);


 //con esto nos aseguramos de que el archivo que se lee tenga la informacion
correspondiente a un tablero
        if (tablero != null && !tablero.isEmpty()) {
            String[] lineas = tablero.split("\n");


            // Extraer el nombre del tablero y las dimensiones
            this.nombreLaberinto = lineas[0];
            String[] informacionDimensiones = lineas[1].split("=");


            //Con el split obtenemos las filas y columnas de este tablero
            String[] dimensiones = informacionDimensiones[1].split("X");
            this.filas = Integer.parseInt(dimensiones[0]);
            this.columnas = Integer.parseInt(dimensiones[1]);


            //Se obtiene el rango de vision del jugador
            String[] sobreLaVision = lineas[2].split("=");
            this.visionDelJugador = Integer.parseInt(sobreLaVision[1]);


            //Se obtien la velocidad que tendran los bots
            String[] sobreElBot = lineas[3].split("=");
            this.velocidadDelBot = Integer.parseInt(sobreElBot[1]);
```

```
//se obtiene la cantidad necesaria de oro para salir del laberinto
String[] SobreLaSalida = lineas[4].split("=");
this.oroParaEscapar = Integer.parseInt(SobreLaSalida[1]);

//para el tablero que no tendra vision
tableroSinVision = new Casilla[filas][columnas];
crearTableroOfuscado();

//Para el tablero que contiene las diferentes casillas
tableroLaberinto = new Casilla[filas][columnas];
panelTablero.setLayout(new GridLayout(filas, columnas));

for (int x = 0; x < filas; x++) {
    String[] caracteres = lineas[x + 5].split(",");
    for (int y = 0; y < columnas; y++) {
        switch (caracteres[y]) {
            case "C":
                tableroLaberinto[x][y] = new Camino();
                break;
            case "P":
                tableroLaberinto[x][y] = new Pared();
                break;
            case "O":
                tableroLaberinto[x][y] = new CaminoOro();
```

```
                    break;
                case "B":
                    Thread botHilo;
                    tableroLaberinto[x][y] = new Bot(this, tableroLaberinto,
panelTablero, filas, columnas, x, y, velocidadDelBot);
                    botHilo = new Thread((Bot) tableroLaberinto[x][y]);

                    //se agrega a la lista de botLista
                    botLista.agregarDato(botHilo);
                    bots.agregarDato((Bot) tableroLaberinto[x][y]);
                    break;
                case "S":
                    tableroLaberinto[x][y] = new Salida(oroParaEscapar);
                    break;
                default:
                    tableroLaberinto[x][y] = new Pared(); // Si el caracter no
coincide, colocamos una pared por defecto
                }
                panelTablero.add(tableroLaberinto[x][y]);
            }
        }
        agregarJugador();
        ventanaJuego.actualizarNombreLaberinto(nombreLaberinto);
        panelTablero.revalidate();
        panelTablero.repaint();
    }
```

```
        System.out.println("Nombre del tablero: " + nombreLaberinto);
        System.out.println("Filas: " + filas);
        System.out.println("Columnas: " + columnas);
        ventanaJuego.mostrarInformacion(oroParaEscapar, bots.obtenerSize(),
velocidadDelBot);

        iniciarBots();
    }
```

EL SIGUIENTE METODO SE ENCARGA DE AGREGAR AL JUGADOR EN UNA CASILLA QUE CONTENGA UN CAMINO. SIGNIFICA QUE ESTA LIBRE Y EL JUGADOR PUEDE SER POSICIONADO ALLI

```
    private void agregarJugador() {
        boolean encontroLugar = false;
        Random random = new Random();
        int rndFila;
        int rndColumna;

        do {
            rndFila = random.nextInt(filas);
            rndColumna = random.nextInt(columnas);

            if (tableroLaberinto[rndFila][rndColumna] instanceof Camino) {
```

```
tableroLaberinto[rndFila][rndColumna].cambiarImagen("/imgCasillas/personaje.png");
                tableroLaberinto[rndFila][rndColumna] = new Jugador(tableroLaberinto, panelTablero, ventanaJuego, filas, columnas, this, visionDelJugador);


                //Para que el tablero conozca al jugador
                jugador = (Jugador) tableroLaberinto[rndFila][rndColumna];
                ventanaJuego.setJugador(jugador);


                posicionX = rndFila;
                posicionY = rndColumna;


                jugador.setPosicionX(posicionX);
                jugador.setPosicionY(posicionY);


                encontroLugar = true;
            }
        } while (!encontroLugar);
    }
```

CON EL SIGUIENTE METODO NOS ENCARGAMOS DE INICIAR A LOS BOTS QUE ESTAN ALMACENADOS EN UN LISTA ENLAZADA

```
  public void iniciarBots() {


      int tamanio = botLista.obtenerSize();
      for (int i = 0; i < tamanio; i++) {
         Thread botActual;
         try {
            botActual = botLista.obtenerValor(i);
            botActual.start();
         } catch (ListaException ex) {
            System.out.println("error 2024");
         } catch (Exception e) {
            System.out.println("Error al iniciar el bot: " + e.getMessage());
         }
      }
   }
```

LA CLASE LISTA ENLAZADA ES OTRA IMPORTANTE YA QUE CON ELLA ALMACENAMOS DIFERENTES DATOS, COMO NUMEROS ENTEROS, LABERINTO, PARTIDA. LA CLASE ESTA PARAMETRIZADA PARA QUE SE PUEDAN CREAR DIFERENTES LISTAS EN BASE A LAS NECESIDADES DEL DESARROLLADOR

CON EL SIGUIENTE METODO SE PUEDEN PAUSAR A LOS BOTS, ESTO SE HACE AL INICIAR UN MINIJUEGO YA QUE LOS BOTS SON HILOS Y SI NO SE PAUSAN ESTOS CONTINUARAN CON SUS MOVIMIENTOS MIENTRAS EL USUARIO ESTA EN EL MINUJUEGO

```
public void pausarBots() {


    int tamanio = bots.obtenerSize();
    for (int i = 0; i < tamanio; i++) {
        Bot botActual;
        try {
            botActual = bots.obtenerValor(i);
            botActual.apagar();
        } catch (ListaException ex) {
            System.out.println("error 2024");
        } catch (Exception e) {
            System.out.println("Error al iniciar el bot: " + e.getMessage());
        }
    }
}
```

CON EL SIGUIENTE METODO LOS BOTS SON PUESTOS EN EJECUCION NUEVAMENTE DESPUES DE TERMINAR UN MINIJUEGO

```
public void prenderBots() {
    int tamanio = bots.obtenerSize();
    for (int i = 0; i < tamanio; i++) {
        Bot botActual;
```

```
        try {
            botActual = bots.obtenerValor(i);
            botActual.prender();
        } catch (ListaException ex) {
            System.out.println("error 2024");
        } catch (Exception e) {
            System.out.println("Error al iniciar el bot: " + e.getMessage());
        }
    }
}
```

CON EL SIGUIENTE METODO PODEMOS FINALIZAR LOS HILOS CUANDO UNA PARTIDA HA LLEGADO A SU FIN

```
public void matarBots() {
    int tamanio = bots.obtenerSize();
    for (int i = 0; i < tamanio; i++) {
        Bot botActual;
        try {
            botActual = bots.obtenerValor(i);
            botActual.finalizarEjecucion();
        } catch (ListaException ex) {
            System.out.println("error 2024");
        } catch (Exception e) {
            System.out.println("Error 2024");
        }
    }
}
```

EL SIGUIENTE METODO SE ENCARGA DE REALIZAR TODAS LAS ACCIONES AL FINALIZAR UNA PARTIDA, COMO CREAR UNA PARTIDA QUE ALMACENARA TODA LA INFORMACION DE LO QUE PASO EN DICHO JUEGO Y ES AGREGADO A UNA LISTA ENLAZADA QUE SE ENCARGA DE LLEVAR UN CONTROL DE TODAS LAS PARTIDAS JUGADAS

```
  public void finalizarPartida() {


      ventanaJuego.terminarElJuegoActual();
      matarBots();


      //Obtener los movimientos para la repeticion
      AccionesEnPartida accionesEnPartida = new
AccionesEnPartida(obtenerDatosBotEnX(), obtenerDatosBotEnY(),
            jugador.getMovimientosEnX(), jugador.getMovimientosEnY(),
            laberintoSeleccionado.getConfiguracionLaberinto());


      //Se debe de guardar aqui todo lo necesario  para los reportes y
repeticiones
      Partidas partida = new Partidas(nombreLaberinto,
jugador.getCantidadOro(), nombreDePartida,
jugador.getCantidadDeMovimientos(),
            listaMovimientoDeLosBots(), cantidadOroWuacamole,
            wuacamoleIniciados, resultadoPartida, verificarSiGano(),
accionesEnPartida);


      archivosPrograma.getListaPartidas().agregarDato(partida);
      archivosPrograma.serializarListas();
  }
```

CON LOS SIGUIETES METODOS SE LOGRA OBTENER TODOS LOS MOVIMIENTOS REALIZADOS POR CADA BOT. ESTO PARA LUEGO UTILIZARLOS EN UNA REPETICION

```
   private ListaGenerica<ListaGenerica> obtenerDatosBotEnX() {
      ListaGenerica<ListaGenerica> listadoDeLosBotsX = new
ListaGenerica<>();

      for (int i = 0; i < bots.obtenerSize(); i++) {
         try {

listadoDeLosBotsX.agregarDato(bots.obtenerValor(i).getMovimientosEnX());
         } catch (ListaException ex) {

Logger.getLogger(ControladorPartida.class.getName()).log(Level.SEVERE,
null, ex);
         }
      }

      return listadoDeLosBotsX;
   }

   private ListaGenerica<ListaGenerica> obtenerDatosBotEnY() {
      ListaGenerica<ListaGenerica> listadoDeLosBotsY = new
ListaGenerica<>();

      for (int i = 0; i < bots.obtenerSize(); i++) {
         try {
```

```
listadoDeLosBotsY.agregarDato(bots.obtenerValor(i).getMovimientosEnY());
        } catch (ListaException ex) {

Logger.getLogger(ControladorPartida.class.getName()).log(Level.SEVERE,
null, ex);
        }
      }

      return listadoDeLosBotsY;
   }
```

LOS SIGUIENTES METODOS SON IMPORTANTES PARA LA PARTE VISUAL DEL JUEGO. YA QUE SE ENCARGAN DE VERIFICAR QUE CASILLAS ESTAN DENTRO DEL RADIO DE VISION DEL JUGADOR Y EN BASE A ESO SE MUESTRAN LAS RESPECTIVAS CASILLAS

```
   public void repintarTablero() {
      panelTablero.removeAll();

      posicionX = jugador.getPosicionX();
      posicionY = jugador.getPosicionY();

      for (int x = 0; x < filas; x++) {
         for (int y = 0; y < columnas; y++) {
            if (esCasillaVisible(posicionX, posicionY, x, y)) {
               panelTablero.add(tableroLaberinto[x][y]);
            } else {
```

```
                panelTablero.add(tableroSinVision[x][y]);
            }
        }
    }
    panelTablero.revalidate();
    panelTablero.repaint();

}
```

SE UTILIZA LA FORMULA DE DISTANCIA DE DOS PUNTOS. ESTO PARA CALCULA QUE CASILLAS ESTAN DENTRO DEL RADIO DE VISION Y CUALES NO

```
/**
 * retorna una posicion valida dentro del rango de vision
 *
 * @param posicionX
 * @param posicionY
 * @param x
 * @param y
 * @return
 */
private boolean esCasillaVisible(int posicionX, int posicionY, int x, int y) {
    //Se obtiene el valor absoluto
    int distanciaX = Math.abs(x - posicionX);
    int distanciaY = Math.abs(y - posicionY);
```

```
        //Con la formula de distancia entre dos puntos
        //d = √(x1-x2)²+(y1-y2)²
        int distanciaXCuadrado = distanciaX * distanciaX;
        int distanciaYCuadrado = distanciaY * distanciaY;

        double distanciaEuclidiana = Math.sqrt(distanciaXCuadrado +
distanciaYCuadrado);

        //Verifica si cumple
        return distanciaEuclidiana < visionDelJugador;
    }
```

SI EL JUGADOR GANA LA PARTIDA SE SUMA UNA VICTORIA AL CONTADO ENCARGADO DE MANEJAR LAS VICTORIAS. ESTO PARA PODER GENEARLO DESPUES EN EL REPORTE GENERAL

```
private int verificarSiGano() {
        if (siGano) {
            return 1;
        } else {
            return 0;
        }
    }
```

CLASE BOT

EN ESTA CLASE SE MANEJA TODO HACERCA DEL BOT. COMO LOS MOVIMIENTOS QUE SON VALIDOS Y CUALES NO. TAMBIEN SI ATRAPA A UN JUGADOR

```
   public Bot(ControladorPartida controladorPartida, Casilla[][]
tableroLaberinto, JPanel panelTablero, int filas, int columnas, int posicionX,
int posicionY, int velocidadBot) {
      super("/imgCasillas/bot.png");


      this.controladorPartida = controladorPartida;
      this.tableroLaberinto = tableroLaberinto;
      this.panelTablero = panelTablero;
      this.filasTablero = filas;
      this.columnasTablero = columnas;
      this.posicionX = posicionX;
      this.posicionY = posicionY;
      this.velocidadDeMovimiento = definirVelocidad(velocidadBot);
      this.estaIniciado = true;


   }
```

ESTA CLASE IMPLEMENTA LA INTERFAZ RUNNABLE, ESTO SE HACE PARA QUE SEA INDEPENDIENTE Y PUEDA REALIZAR CIERTA CANTIDAD DE MOVIMIENTOS EN UN SEGUNDO

```
public void run() {
    Random random = new CRandom();
    while (true) {
      if (estaIniciado) {
        int movimiento = random.nextInt(4);

        // Realizar un movimiento aleatorio
        switch (movimiento) {
          case 0:
            cambiarPosicion(posicionX, posicionY - 1);
            break;

          case 1:
            cambiarPosicion(posicionX, posicionY + 1);
            break;

          case 2:
            cambiarPosicion(posicionX - 1, posicionY);
            break;

          case 3:
            cambiarPosicion(posicionX + 1, posicionY);
            break;
```

```
                default:
                    break;
            }

            try {
                Thread.sleep(velocidadDeMovimiento); //Se duerme el hilo en base a lo definido en el constructor de laberintos
            } catch (InterruptedException e) {
                // Manejar la interrupción si es necesario
                System.out.println("algo paso.. ver si llega hasta aqui");
                //e.printStackTrace();
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("ErrorCapturrado 2404");
                //e.printStackTrace();
            }
        } else {
            try {
                Thread.sleep(100);  // Si está pausado, esperar antes de volver a verificar
            } catch (InterruptedException e) {
                System.out.println("El hilo del bot ha sido interrumpido mientras está pausado.");
            }
        }
        if (terminado) {
```

```
            break;
        }
    }
    {
    }
    System.out.println("el hilo bot ha finalizado");
  }
```

ESPERA CIERTA CANTIDAD DE MILISEGUNDOS ENTRE CADA MOVIMIENTO, ESTO EN BASE A LA DECISION DEL USUARIO QUE ELIJE LA VELOCIDAD DEL BOT

CUANDO UNA PARTIDA LLEGA A SU FIN ESTE ES EL METODO QUE SE EJECUTA EN LA CLASE BOT PARA FINALIZAR EL HILO

```
public void finalizarEjecucion() {
    terminado = true;
  }
```

EN BASE A UN NUMERO ALEATORIO ELEJIDO POR LA COMPUTADORA EL BOT ELIJE UNA DIRECCION HACIA DONDE MOVERSE CON EL SIGUIENTE METODO

```
    //guarda todos los movimientos hechos para luego utilizarlos en la repeticion
    movimientosEnX.agregarDato(x);
    movimientosEnY.agregarDato(y);
```

```
        if (x < 0 || x >= filasTablero || y < 0 || y >= columnasTablero) {
            System.out.println("Movimiento inválido: coordenadas fuera de los límites del tablero");
            return;
        }
        if (tableroLaberinto[x][y] instanceof Camino) {
            //si es camino debe avanzar
            avanceBot(x, y);
            repintar();
        } else if (tableroLaberinto[x][y] instanceof CaminoOro) {
            //si se encuentra con una casilla que contiene oro
            avanceBot(x, y);
            repintar();
        } else if (tableroLaberinto[x][y] instanceof Jugador) {

            //Para mostrar la animacion correctamente
            this.cambiarImagen(imagenCamino);
            //si el bot atrapa  al jugador se realizan lo programado en el metodo de la clas Jugador
            Jugador player = (Jugador) tableroLaberinto[x][y];
            tableroLaberinto[x][y].realizarAccionCasilla(player);
        } else {
            //No hacer ningun Movimiento
        }
    }
```

CUANDO UN MOVIMIENTO ELEGIDO POR EL BOT ES VALIDO SE EJECUTA LO SIGUIENTE

```
private void avanceBot(int x, int y) {
    cantidadMovimientos++;
    devolverAnterior();
    guardarPaso = tableroLaberinto[x][y];
    verificarDireccion(x, y);
    tableroLaberinto[posicionX][posicionY] = this;

}
```

SE VERIFICA LA DIRECCION QUE TOMO EL BOT CON EL SIGUIENTE METODO

```
private void verificarDireccion(int x, int y) {
    if (x == posicionX - 1) {
        posicionX--;
    } else if (x == posicionX + 1) {
        posicionX++;
    } else if (y == posicionY - 1) {
        posicionY--;
    } else if (y == posicionY + 1) {
        posicionY++;
    }
}
```

Y SE DEVUELVE LA CASILLA PISADA ANTERIORMETE POR EL BOT CON EL SIGUIENTE METODO

```
  private void devolverAnterior() {
      tableroLaberinto[posicionX][posicionY] = guardarPaso;
   }
```

CUANDO ALGUIEN SE POSICIONA EN UNA CASILLA QUE ES UN BOT SE EJECUTA EL SIGUIENTE METODO

```
@Override
   public void realizarAccionCasilla(Jugador jugador) {
      //Si se paran en esta casilla
      //Avisar que se perdio al frontend
      jugador.cambiarImagen(imagenCamino);
      jugador.getControladorPartida().matarBots();
      JOptionPane.showMessageDialog(null, "Has perdido, Te atrapo un bot");
      jugador.getControladorPartida().setResultadoPartida("Perdio");

jugador.getControladorPartida().getLaberintoSeleccionado().aumentarVecesP
erdido();


      jugador.getControladorPartida().finalizarPartida();
   }
```

CLASE JUGADOR

EN ESTA CLASE SE REALIZAN TODAS LAS ACCIONES DEL JUGADOR, PERO ESTO SON REALIZADOS ATRAVES DE LA VENTANA JUEGO QUE TIENE UN KEYLISTENER PARA CAPTUAR LOS MOVIMIENTOS DEL JUGADOR A,D,S,W.

```
@Override
public void keyReleased(KeyEvent e) {
    int keyCode = e.getKeyCode();
    switch (keyCode) {
        case KeyEvent.VK_W:
            jugador.realizarMovimiento("w");
            break;
        case KeyEvent.VK_A:
            jugador.realizarMovimiento("a");
            break;
        case KeyEvent.VK_S:
            jugador.realizarMovimiento("s");
            break;
        case KeyEvent.VK_D:
            jugador.realizarMovimiento("d");
            break;
        default:
            break;
    }
```

CADA TECLA DETECTADA REALIZA UN MOVIMIENTO EN LA DIRECCION INDICADA DENTRO DEL METODO REALIZARMOVIMIENTO

UNA DE LAS CLASE MAS IMPORTANTES ES LA DE ARCHIVOSPROGRAMA, YA QUE ESTA CLASE SE ENCARGA DE ALMACENAR TODA LA INFORMACION DE LAS PARTIDAS DEL USUARIO DESDE SU PRIMERA EJECUCION. EN DONDE SE CREA UN ARCHIVO CON UN PATH YA DEFINIDO DENTRO DE LA CLASE Y SE LE AGREGAN DOS LISTAS ENLAZADA, UNA CON TODA LA INFORMACION DE LOS LABERINTOS Y LA OTRA CON TODA LA INFORMACION DE LAS PARTIDA

```
private static final String PATH_ARCHIVO = "RegistroJuego.dat";
private static final String PATH_REPORTE = "Reporte.html";
```

EN EL CONSTRUCTOR SE INICIALIZAN LAS LISTA CADA VEZ QUE ESTA ES INSTANCIADA

```
public ArchivosPrograma() {
    //Al instanciar a la clase se tiene que verificar si ya existe un archivo o no
    verificarArchivosJuego();
}
```

CON EL SIGUIENTE METODO SE ALMACENAN LAS LISTAS EN UN ARCHIVO BINARIO QUE CONTENDRA TODA LA INFORMACION

```
public void serializarListas() {
    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(PATH_ARCHIVO))) {
        oos.writeObject(listaLaberintos);
        oos.writeObject(listaPartidas);

        System.out.println("Lista serializada con éxito.");
```

```
        } catch (IOException e) {
            System.err.println("Error al serializar la lista: " + e.getMessage());
        }
    }
```

CON EL SIGUIENTE METODO SE DESERIALIZAN AMBAS LISTA SI ES QUE EXISTEN Y ASI SE RECUPERA EL PROGRESO DEL USUARIO

```
    public void deserializarListas() {
        try (ObjectInputStream oisRegistros = new ObjectInputStream(new
FileInputStream(PATH_ARCHIVO))) {

            Object objLaberintos = oisRegistros.readObject();
            Object objPartidas = oisRegistros.readObject();

            //verifica que el objeto deserealizado si sea una instancia de
ListaGenerica
            //Se le agrega el valor de la lista guardada a la variable local
            listaLaberintos = (ListaGenerica<Laberinto>) objLaberintos;
            listaPartidas = (ListaGenerica<Partidas>) objPartidas;

            System.out.println("Lista deserializada con éxito.");

        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error al deserializar la lista: " + e.getMessage());
        }
    }
```

CON EL SIGUIENTE METODO SE VERIFIACA SI YA EXISTE EL ARCHIVO QUE CONTIENE TODA LA INFORMACION DEL JUEGO, SI EXISTE DESERIALIZA LAS LISTAS, SI NO EXISTE CREA UN NUEVO ARCHIVO Y PROCEDE A INICIALIZAR AMBAS LISTA AGREGADO EL TABLERO POR DEFECTO A LA LISTA DE LABERINTOS

```
   private void verificarArchivosJuego() {
      File archivo = new File(PATH_ARCHIVO);


      if (archivo.exists()) {
         // Si el archivo existe, deserializar la lista de tableros
         deserializarListas();
      } else {
         // Si el archivo no existe, crear uno nuevo y agregar el tablero por defecto
         System.out.println("Se ha creado el tablero por defecto");


         //Se inicializa la lista de tableros
         listaLaberintos = new ListaGenerica<>();
         // Agregar el tablero por defecto a la lista
         Laberinto tableroDefecto = new Laberinto(laberintoPorDefecto);
         listaLaberintos.agregarDato(tableroDefecto);


         System.out.println("Se ha creado una nueva lista de partidas");
         //Se inicializa la lista
         listaPartidas = new ListaGenerica<>();
```

```
            // Serializar la lista de tableros al nuevo archivo
            serializarListas();
        }
    }
```

CON EL SIGUIENTE METODO SE PUEDEN EXPORTAR DIFERENTES TIPOS DE REPORTE A HTML, EL USUARIO SOLAMENTE DEBE DE INDICAR QUE REPORTE QUIERE EXPORAR

```
  public void exportarReporteHTML(String Contenido) {
        File reporte = new File(PATH_REPORTE);


        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(PATH_REPORTE))) {


            //Si no existe el archivo lo crea
            if (!reporte.exists()) {
                reporte.createNewFile();  // Crea el archivo si no existe
            }


            StringBuilder builder = new StringBuilder();
            String[] lineas = Contenido.split("\n");
            for (int i = 0; i < lineas.length; i++) {

builder.append("<p>").append(lineas[i]).append("</p>").append("\n");
            }
```

```
            bw.write("<html>" + "\n"
                    + " <head>" + "\n"
                    + "     <title>Reporte Partida</title>" + "\n"
                    + " </head>" + "\n"
                    + "     <body>" + "\n"
                    + builder.toString()
                    + "     </body>" + "\n"
                    + "</html>");


            JOptionPane.showMessageDialog(null, "Reporte exportado
exitosamente");


        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error al exportar el reporte");
        }
    }
```

CON EL SIGUIETE METODO SE PUEDE AGREGAR UN LABERINTO NUEVO A LA LISTA

```
public void agregarUnLaberinto(Laberinto laberinto) {
        listaLaberintos.agregarDato(laberinto);
        System.out.println("laberinto agregado exitosamente");
    }
```