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 PRACTICA 2”

Melvin Eduardo Ordoñez Sapón RA | 202230552

Quetzaltenango, abril del 2024

“Id y enseñad a todos”

ACERCA DEL PROGRAMA:

IDE utilizado: Apache NetBeans IDE 20

JDK utilizado: Versión 21.0.2.0

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 METODO ES UNO DE LOS PRINCIPALES PARA INICIAR UNA PARTIDA YA QUE ES EL CONTRUCTOR DEL LA CLASE MOTOR.

```
public MotorJuego(VentanaJuego framePrincipal, GeneracionDeTablero
tableroGenerado) {
    //esta parte del constructor obtiene todo lo que necesita para iniciar con
la partida
    this.framePrincipal = framePrincipal;
    this.tableroEnJuego = tableroGenerado;
    this.botones = tableroGenerado.getBotones();
    this.filas = tableroGenerado.getFilas();
    this.columnas = tableroGenerado.getColumnas();
    this.cantidadParejasGeneradas = (columnas * filas) / 2;

    JOptionPane.showMessageDialog(null, "Se ha empezado una nueva
partida");
    crearJugadores();
```

```
        elegirQuienEmpieza();
        actualizarEstadisticas();
        controladorPartida();
    }
```

ESTE METODO SE ENCARGA DE CREAR A DOS JUGADORES Y SE LOS ASIGNA AL MOTOR DEL JUEGO PARA QUE INICIE

```
public void crearJugadores() {

        PedirNombre pedirName = new PedirNombre(framePrincipal, true);
        pedirName.setVisible(true);
        jugador1 = new Jugador(pedirName.getNombreCapturado());

        PedirNombre pedirName2 = new PedirNombre(framePrincipal, true);
        pedirName2.setVisible(true);
        jugador2 = new Jugador(pedirName2.getNombreCapturado());

    }
```

ESTE METODO SE ENCARGA DE ELEGIR ALEATORIAMENTE A UN JUGADOR Y SE LO ASIGNA A LA VARIABLE QUE GUARDA AL JUGADOR EN TURNO

```
private void elegirQuienEmpieza() {
        Random random = new Random();
        int numrandom = random.nextInt(2);

        switch (numrandom) {
            case 0:
                jugadorEnTurno = jugador1;
                break;

            case 1:
                jugadorEnTurno = jugador2;
                break;

            default:
                //no deberia de llegar a este punto, pero por si al caso
                JOptionPane.showMessageDialog(null, "ha ocurrido un error
inesperado");
```

```
    }
  }
```

ESTE METODO SE ENCARGA DE AGREGARLE EL EVENTO DEL CLIK A UN BOTON Y ESE MISMO EVENTO TIENE OTRAS PROPIEDADES DE LA CLASE MOTOR JUEGO QUE SIRVE PARA LLEVAR UN CONTROL DE LAS CARTAS Y TURNOS

```
private void funcionBotones() {
      for (int x = 0; x < filas; x++) {
         for (int y = 0; y < columnas; y++) {
            final int filaActual = x;
            final int columnaActual = y;

            botones[x][y].addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {
                  //Agregar Funciones a la hora de Presionar un boton
                  botones[filaActual][columnaActual].mostrarId();
                  controlarPresiones(botones[filaActual][columnaActual]);
                  if (finalPartida()) {
                     definirGanador();
                     framePrincipal.dispose();
```

```
                    MenuPrincipal menuPrincipal = new MenuPrincipal();
                    menuPrincipal.setVisible(true);
                    //MostrarGanadores

                }
            }
        });
    }
}
```

ESTE METODO SE ENCARGA DE VERIFICAR DOS CARTAS SELECCIONADAS POR EL JUGADOR EN TURNO Y HACE UNA COMPROBACION A BASE DEL ID QUE CONTIENEN LAS CARTAS, ESO FUNCIONA COMO IDENTIFICADOR

```
private void verificarPareja() {

    if (cartaPresionada1.getId() == cartaPresionada2.getId()) {
        /**
         * si el jugador acerto un par de cartas este sigue en turno y se
```

```
         * realizan las respectivas acciones por conseguir una pareja
         */

        cartaPresionada1.setEnabled(false);
        cartaPresionada2.setEnabled(false);

        cartaPresionada1.setBackground(Color.CYAN);
        cartaPresionada2.setBackground(Color.CYAN);

        parejaEncontrada = true;
        cantidadParejasGeneradas--;

        //Como el jugador en turno encontro una pareja se le suman los
puntos
        jugadorEnTurno.sumarPuntos();
        actualizarEstadisticas();

        if (finalPartida()) {
            JOptionPane.showMessageDialog(null, "Ya se han encontrado
todas las parejas");
```

```
    }

} else {
    /**
     * Si no se encuentra una pareja se restablecen las cartas que no
     * coincidieron y realiza las respectivas acciones al jugador por
     * fallar
     */
    jugadorEnTurno.restarPuntos();
    actualizarEstadisticas();
    cambiarTurno();
    cartaPresionada1.setText("");
    cartaPresionada2.setText("");
    cartaPresionada1.setIcon(null);
    cartaPresionada2.setIcon(null);

    cartaPresionada1.setEnabled(true);
    cartaPresionada2.setEnabled(true);
    parejaEncontrada = false;
}
```

ESTE METODO SIRVE PARA VER SI HAY UN FINAL DE PARTIDA Y SI ES ASI EL CONTROLADOR REALIZARA LAS ACCIONES DE FINALIZACION DE PARTIDA

```
private boolean finalPartida() {
        return cantidadParejasGeneradas == 0;
    }
```

ESTE METODO SE ENCARGA DE GUARDAR A LOS JUGADORES GANADORES EN UN ARCHIVO DE TEXTO YA CON UN NOMBRE DEFINIDO, EL PATH_ARCHIVO

```
    public void guardarPartidaGanada(Jugador jugador) {
        File archivo = new File(PATH_ARCHIVO);
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(archivo, true))) {
            if (!archivo.exists()) {
                archivo.createNewFile();  // Crea el archivo si no existe
            }

            writer.write(jugador.getNombre() + "," + jugador.getPunteo() + ","
```

```
                + jugador.getHoraDeJuego() + "," +
jugador.getNombrePerdedor() + "\n");
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Error al guardar la partida
ganada");
        }
    }
```

ESTE METODO ES IMPORTANTE PARA PODER ACCEDER A LAS ESTADISTICAS TOTALES DE LOS JUGADORES, FUNCIONA LEYENDO EL ARCHIVO DE TEXTO

```
  public Jugador[] leerArchivo() {
        Jugador[] players = null;

        try (BufferedReader reader = new BufferedReader(new
FileReader(PATH_ARCHIVO))) {
            String linea;
            int contLine = 0;
            while ((linea = reader.readLine()) != null) {
                contLine++;
```

```
        }

        // Crear un nuevo BufferedReader para leer desde el principio del
archivo
        try (BufferedReader newReader = new BufferedReader(new
FileReader(PATH_ARCHIVO))) {
            players = new Jugador[contLine];
            int contadorJugadores = 0;
            while ((linea = newReader.readLine()) != null) {
                String[] data = linea.split(",");
                //SE HACE EL CASTEO DE CADA JUGADOR Y SE LE
PASA A UN CONSTRUCTOR ESPEFICICO
                //QUE SE ENCARGA DE RESTABLECER ESOS DATOS
                LocalDateTime hora = LocalDateTime.parse(data[2]);
                Jugador nuevo = new Jugador(data[0], Integer.valueOf(data[1]),
hora, data[3]);
                players[contadorJugadores] = nuevo;
                contadorJugadores++;
            }
        }
```

```
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "No hay un historial de
jugadores");
        }
        return players;
    }
```