MANUAL TECNICO

SNAKE GAME

Luis Andres Cobar Sandoval
INGENIERIA EN CIENCIAS Y SISTEMAS

INTRODUCCION

El Programa SnakeGame es un programa donde se utilizan hilos para realizar varios procesos al mismo tiempo que sean distintos al hilo principal.

OBJETIVOS

El objetivo primordial de este manual es ayudar a los distintos programadores y aspirantes al conocimiento de las ciencias de la computación así mismo del funcionamiento de los Hilos de procesos.

Ventana Design: Se colocan los parámetros para la ventana principal.

```
this.setTitle("Snake");
        this.setLayout(null);
        this.setBounds(0, 0, 900, 660);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.getContentPane().setBackground(gray);
        this.setVisible(true);

        setLocationRelativeTo(null);
```

Selector de dificultades: Se coloca el diseño del JcomboBox

```
JComboBox Dificultades = new JComboBox();
Dificultades.setForeground(BLACK);
Dificultades.setBackground(WHITE);
Dificultades.setBorder(null);
Dificultades.setBounds(614, 250, 265, 80);
Dificultades.setFont(new Font("Ubuntu", Font.PLAIN, 26));
Dificultades.addItem("Facil 😅");
Dificultades.addItem("Normal 😀");
Dificultades.addItem("Dificil 🤕");
Dificultades.setSelectedItem("Normal 😀");
Dificultades.setRenderer(new DefaultListCellRenderer() {
    @Override
    public void paint(Graphics g) {
        setBackground(Color.WHITE);
        setForeground(Color.BLACK);
        setBorder(null);
        super.paint(g);
        setHorizontalAlignment(CENTER);
    }
});
this.add(Dificultades);
Dificultades.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (Dificultades.getSelectedItem() == "Facil 😅") {
            dificultad = 0;
        } else if (Dificultades.getSelectedItem() == "Normal 😀") {
            dificultad = 1;
        } else if (Dificultades.getSelectedItem() == "Dificil 🤕") {
            dificultad = 2;
        }
    }
});
```

Creacion de Hilo de tiempo: Se crea un nuevo hilo, así como también se le da los parámetros Swing

```
this.Time = new Tiempo();
this.tiempo_consumir = new Thread(this.Time);
Time.setBounds(614, 10, 265, 80);
Time.setHorizontalAlignment((int) CENTER_ALIGNMENT);
Time.setBackground(BLACK);
Time.setOpaque(true);
Time.setForeground(WHITE);
this.add(Time);
```

Panel Snake: Se inicializa el Panel de Snake, en el cual se crea el personaje Snake, la comida y la matriz en la cual se movera.

```
Snake = new PanelSnake(600, 15);
this.add(Snake);
Snake.setBounds(6, 10, 600, 600);
Snake.setOpaque(false);
```

Panel Tablero: Se inicializa el tablero el cual dará la imagen de fondo de donde recorrerá la serpiente.

```
fondo = new PanelJ(600, 15);
   this.add(fondo);
   fondo.setBounds(6, 10, 600, 600);
```

Boton Generar Top15: Se crea el botón para generar el top 15, en el cual se le dan sus atributos visuales así como también inicializa el método generarTop()

```
JButton TOP15 = new JButton("🏆  TOP 15  🏆");
       TOP15.setBounds(614, 430, 265, 80);
       TOP15.setBackground(white);
       TOP15.setForeground(BLACK);
       TOP15.setFont(new Font("Ubuntu", Font.BOLD, 30));
       TOP15.setBorder(null);
       TOP15.setVisible(true);
       this.add(TOP15);
       TOP15.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               generarTop();
           }
       });
       TOP15.addMouseListener(new MouseAdapter() {

           public void mouseEntered(MouseEvent evt) {
               TOP15.setBackground(lightGray);
           }

           public void mouseExited(MouseEvent evt) {
               TOP15.setBackground(white);
           }
       });
```

Boton Iniciar: Se crea el botón para iniciar, se le colocan sus atributos visuales y cuando se activa inicia el proceso de Snake.Iniciar(), se pide que el programa se enfoque en los movimientos que se harán, se crea un nuevo hilo de tiempo y se reiniciar el JLabel de tiempo, y se desactiva el botón de iniciar.

```
// ===== Iniciar Boton =====
        Inciar = new JButton("INICIAR");
        Inciar.setBounds(614, 90, 265, 80);
        Inciar.setBackground(white);
        Inciar.setForeground(BLACK);
        Inciar.setFont(new Font("Ubuntu", Font.BOLD, 30));
        Inciar.setBorder(null);
        Inciar.setVisible(true);
        Inciar.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent evt) {
                Inciar.setBackground(lightGray);
            }

            public void mouseExited(MouseEvent evt) {
                Inciar.setBackground(white);
            }
        });
        Inciar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (Repit == false) {
                    Snake.iniciar(true, dificultad);
                    requestFocus();
                    tiempo_consumir = new Thread(Time);
                    Time.reiniciar();
                    tiempo_consumir.start();
                    Inciar.setEnabled(false);
                }
            }
        });
        this.add(Inciar);
```

Snake.Iniciar(): Se inicializa el constructor del panel en el cual se encuentra el personaje de Snake así como también se le otorgan sus atributos.

```
public void iniciar(boolean _Iniciar, int _Dificultad){
        if(_Iniciar == true){
            int[] a = {SizePanelM / 2 - 1, SizePanelM / 2 - 1};
            int[] b = {SizePanelM / 2, SizePanelM / 2 - 1};
            Snake.add(a);
            Snake.add(b);
            Caminar = new HiloCaminar(this);
            CaminarH = new Thread(Caminar);
            Caminar.Dificult(_Dificultad);
            Pintar =true;
            generarComida();
            repaint();
            CaminarH.start();
        }
    }
```

Boton de Reinicio: Se agrega un botón de reinicio con sus atributos visuales, así como también un addActionListener.

```
// ===== Agregar Boton Reiniciar =====
      Reiniciar = new JButton("REINTENTAR");
      Reiniciar.setBounds(614, 170, 265, 80);
      Reiniciar.setVisible(true);
      Reiniciar.setEnabled(false);
      Reiniciar.addActionListener(this);
      Reiniciar.setBackground(white);
      Reiniciar.setForeground(BLACK);
      Reiniciar.setFont(new Font("Ubuntu", Font.BOLD, 30));
      Reiniciar.setBorder(null);
      Reiniciar.addMouseListener(new MouseAdapter() {

          public void mouseEntered(MouseEvent evt) {
              Reiniciar.setBackground(lightGray);
          }

          public void mouseExited(MouseEvent evt) {
              Reiniciar.setBackground(white);
          }
      });
      this.add(Reiniciar);
```

El cual reinicia los hilos así como también reinicializa el JPanel de Snake.

```
if (e.getSource() == Reiniciar) {
            this.remove(Snake);
            this.remove(fondo);
            this.remove(Time);
            Time = new Tiempo();
            this.add(Time);
            repaint();
            Time.setVisible(true);
            Time.setBounds(614, 10, 265, 80);
            Time.setHorizontalAlignment((int) CENTER_ALIGNMENT);
            Time.setBackground(BLACK);
            Time.setOpaque(true);
            Time.setForeground(WHITE);
            PanelSnake.puntos = 0;
            fondo = new PanelJ(600, 15);
            fondo.setBounds(6, 10, 600, 600);
            Snake = new PanelSnake(600, 15);
            Snake.setBounds(6, 10, 600, 600);
            Snake.setOpaque(false);
            this.add(Snake);
            Snake.setVisible(true);
            this.add(fondo);
            Reiniciar.setEnabled(false);
            Inciar.setEnabled(true);
            Puntaje.setText("Puntos:  /25");
            tiempo_consumir.interrupt();
        }
```

Detector de Evento: Detecta la tecla presionada para saber a que dirección debe moverse

```
    // ==== Detector de Tecla Presionada ====
        this.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent evt) {
                formKeyPressed(evt);
            }
        });
        this.requestFocus(true);
    }
```

```
private void formKeyPressed(KeyEvent evt) {
        switch (evt.getKeyCode()) {
            case KeyEvent.VK_LEFT:
                Snake.CambiarDireccion("Left");
                break;
            case KeyEvent.VK_UP:
                Snake.CambiarDireccion("Up");
                break;
            case KeyEvent.VK_RIGHT:
                Snake.CambiarDireccion("Right");
                break;
            case KeyEvent.VK_DOWN:
                Snake.CambiarDireccion("Down");
                break;
        }
    }
```

Finalizar(): Método el cual finaliza la cuenta de tiempo en el hilo de tiempo.

```
public static void Finalizar() {
        Time.stop();
    }
```

fin(): el cual es llamado cada vez que el jugador pierde o gana para crear los datos del objeto Puntaciones.

```
public static void fin(){
        Puntuaciones c = new Puntuaciones(PanelSnake.puntos,Time.getText());
        Principal.generarPuntaje(c);
    }
```

Además de ello llama al método de **generarPuntaje()**

generarPuntaje(): el cual ingresa los datos en un array de objetos.

```
public static void generarPuntaje(Puntuaciones c){
        if(contadorTop <15){
            top15[contadorTop] = c;
            contadorTop++;
        }
    }
```

generarTop(): En el cual se crea un nuevo array en el cual ingresaremos los objetos Puntaciones, con la longitud de intentos realizados, este se ordena y se manda como un string al método generarReporte()

```
public static void generarTop() {
        System.out.println();
        String texto = "";
        System.out.println(" ======= TOP 15 ======");
        Puntuaciones[] arreglotemporal = new Puntuaciones[contadorTop];
        for (int i = 0; i < contadorTop; i++) {
            arreglotemporal[i] = top15[i];
        }
        Arrays.sort(arreglotemporal);

        for (int i = 0; i < contadorTop; i++) {
            System.out.println(arreglotemporal[i].getPuntuacion() + " " + arreglotemporal[i].getTiempo());
            texto = texto + "<p>"+(i+1) + ". Puntuacion: " + arreglotemporal[i].getPuntuacion() + " Tiempo: "
+ arreglotemporal[i].getTiempo() + "</p>\n";
        }
        System.out.println(" ====================");
        System.out.println();
    htmlGenerator.generarReporte(texto);
    }
```

generarReporte(): en el cual inicializa 3 metodos mas además de atribuir el String enviado desde el método generarTop() al método de inicioHTML()

```
public static void generarReporte(String texto){
        inicioHTML(texto);
        crearReporte();
        abrirReporte();
    }
```

inicioHTML(): El método genera el texto del archivo HTML.

```
public static void inicioHTML(String texto){
        textohtml.append("<!doctype html>\n" +
                "<html lang=\"en\">\n"+
                "<head>\n"+
                "<meta charset=\"UTF-8\">"+
                "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">"+
                "<link rel=\"stylesheet\" href=\"./style.css\">"+
                "<title>Mejores Puntaciones Snake</title>"+
                "</head>\n"+
                "<header>\n"+
                "<img src=\"./images/Serpiente.png\" alt=\"Andres Cobar\">\n"+
               "<section>\n"+
                "<h1>Top Puntuaciones Snake</h1>\n"+
                "</section>  \n"  +
                "<div class=\"spacer\">\n"+
                "<img src=\"./images/fiusac_negro.png\" alt=\"Andres Cobar\">\n"+
                "</div>\n"+
                "</header>\n"+
                "<body>\n"+
                "<div class = \"caja\">\n"+
                "<div class=\"top\">\n"+
                "<h1> 15</h1>\n"
        );

            textohtml.append(texto);
        textohtml.append("</div>\n" +
                "</div>\n" +
                "</body>\n");

    }
```

abrirReporte(): Indicamos donde estará el archivo así como también se crea el mensaje de reporte generado.

```
public static void abrirReporte(){
        try
        {
            File file = new File("./Reportes/Reporte_Errores.html");
            if(!Desktop.isDesktopSupported())
            {
                System.out.println("not supported");
                return;
            }
            Desktop desktop = Desktop.getDesktop();
            if(file.exists())
                desktop.open(file);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
```

Avanzar(): El cual genera el movimiento de matriz según la dirección, así como también comprueba que si esta tocando con la comida para sumar puntos y para comprobar si se ha ganado.

```
public void Avanzar() {
        IgualarDireccion();
        int[] ultimo = Snake.get(Snake.size() - 1);
        int MoveX = 0;
        int MoveY = 0;
        switch (Direccion) {
            case "Right":
                MoveX = 1;
                break;
            case "Left":
                MoveX = -1;
                break;
            case "Up":
                MoveY = -1;
                break;
            case "Down":
                MoveY = 1;
                break;
        }
        int[] nuevo = {Math.floorMod(ultimo[0] + MoveX, SizePanelM), Math.floorMod(ultimo[1] +
MoveY, SizePanelM)};
        boolean  existe = false;
        for (int i = 0; i < Snake.size(); i++){
            if(nuevo[0] == Snake.get(i)[0] && nuevo[1] == Snake.get(i)[1]){
                existe = true;
                break;
            }
        }

        if(existe){
            tiempof = Principal.Time.getText();
            Principal.Finalizar();
            Caminar.stop();

            JOptionPane.showMessageDialog(null,"Perdiste");
            Principal.fin();
            Principal.Reiniciar.setEnabled(true);
            Principal.Inciar.setEnabled(false);

        }else {
            if(nuevo[0] == comida[0] && nuevo[1] == comida[1]){
                puntos++;
                Snake.add(nuevo);
                generarComida();
                Principal.Puntaje.setText("Puntos: " + puntos+"/25");
                if (puntos == 25){
                    JOptionPane.showMessageDialog(null,"🎉 ¡¡¡HAZ GANADO!!! 🎉");
                    tiempof = Principal.Time.getText();
                    Principal.fin();
                    Principal.Finalizar();
                    Caminar.stop();
                    Principal.Reiniciar.setEnabled(true);
                    Principal.Inciar.setEnabled(false);
                }
            }else{
                Snake.add(nuevo);
                Snake.remove(0);
            }
        }

    }
        public void generarComida() {
            boolean Exist = false;
            int a = (int) (Math.random() * SizePanelM);
            int b = (int) (Math.random() * SizePanelM);
            for (int[] par : Snake) {
                if (par[0] == a && par[1] == b) {
                    Exist = true;
                    break;
                }
            }
            if(!Exist){
                this.comida[0] = a;
                this.comida[1] = b;
            }
        }
```

generarComida(): crea 2 números aleatorios para ingresarlos en una matriz para generar la comida, así como también comprueba que no sea en un punto donde esta Snake

```
public void generarComida() {
        boolean Exist = false;
        int a = (int) (Math.random() * SizePanelM);
        int b = (int) (Math.random() * SizePanelM);
        for (int[] par : Snake) {
            if (par[0] == a && par[1] == b) {
                Exist = true;
                break;
            }
        }
        if(!Exist){
            this.comida[0] = a;
            this.comida[1] = b;
        }
    }
```