UNIVERSIDAD DE MARGARITA

ALMA MATER DEL CARIBE

VICERRECTORADO ACÁDEMICO

DECANATO DE INGENIERIA EN SISTEMAS

UNIDAD CURRICULAR: EDD

SECCION: S-01

JUEGO DE MEMORIA

Facilitador

Cesar Requena

Autor:

Geyser Velásquez C.I 31.369.312

Manuel Cacique C.I 29.864.784

El Valle del Espíritu Santo, 23 de Octubre de 2023

Análisis del Problema

En el proyecto propuesto en el curso de estructura de datos, se nos requirió para desarrollar un programa que modele un juego de destreza mental, es decir, un juego de memoria. Para lo cual hicimos uso del lenguaje de programación orientado a objetos Java, y de la IDE Neat Beans. Durante el desarrollo de este proyecto primero definimos la estructura de datos a utilizar, primero pensamos en un arreglo bidimensional de enteros, en el cual se iban a registrar números correspondientes a cada carta del juego. Posteriormente nos percatamos que no era necesaria una estructura bidimensional, ya que podíamos hacer uso de una estructura unidimensional y reflejar sus elementos en la interfaz gráfica de forma tal que su representación gráfica tuviera aspecto de una matriz.

Uno de los requerimientos era el que pudiera jugar mínimo 2 jugadores, para lo cual nos surgió la interrogante de como estructurar el paso de los turnos. Pensamos en usar una lista circular, para que se repitieran constantemente los turnos hasta que en una condición rompiera el bucle de la lista circular. Pero debido a que fijamos el número de jugadores en 2, esta idea fue descartada e implementamos una idea más sencilla. Mediante orientación a objetos creamos algoritmos que nos permitieran ir pasando los turnos de jugador a jugador.

Para el contenido de las cartas en un principio usamos un formato binario, 0s y 1s, para hacer pruebas en la consola e ir familiarizando con Java e ir ideando la lógica que implementaríamos en la interfaz gráfica. Debido a los resultados obtenidos en esta fase decidimos optar por un sistema de juego, en el cual, la forma de identificar cuando un jugador escogió pares exitosos se realiza mediante identificadores numéricos en las cartas; si estos identificadores coinciden , entonces se escogió un par exitoso. Al escoger dos cartas con el mismo ID, el jugador en cuestión obtiene una cierta cantidad de puntos. Además, para evitar en la mayor medida los empates, a los puntos obtenidos se le restan la cantidad de intentos que realizaron cada jugador.

Uno de los aspectos a los que más importancia le dimos fue al apartado estético. Se discutió mucho sobre que figuras iban a ser usadas para el juego. Pasamos de números a colores, de colores a figuras geométricas y de figuras geométricas a algo más concreto, que termino siendo sprites de la famosa saga de videojuego Pokemon. Se creo un fondo para cada carta, con un signo de interrogación para cuando estuvieran ocultas y con su respectivo Sprite para cada carta cuando estuvieran volteadas.

Uno de los aspectos mas resaltantes en el código es una estructura repetitiva del tipo For, la cual itera 9 veces, que contiene otra estructura for, la cual itera 2 veces. Dando un total de 18 iteraciones, es decir, 9 veces 2 veces. Esto con la obvia función de generar varios pares de cartas con los mismos atributos sin la necesidad de tener que hacer uso de código repetitivo. Esta estructura fue parte fundamental para llevar a cabo una solución al problema que se nos había propuesto, debido a que dentro de ella se realizan tanto la creación de las cartas como su añadimiento a la lista ‘’MazoCartas’’.

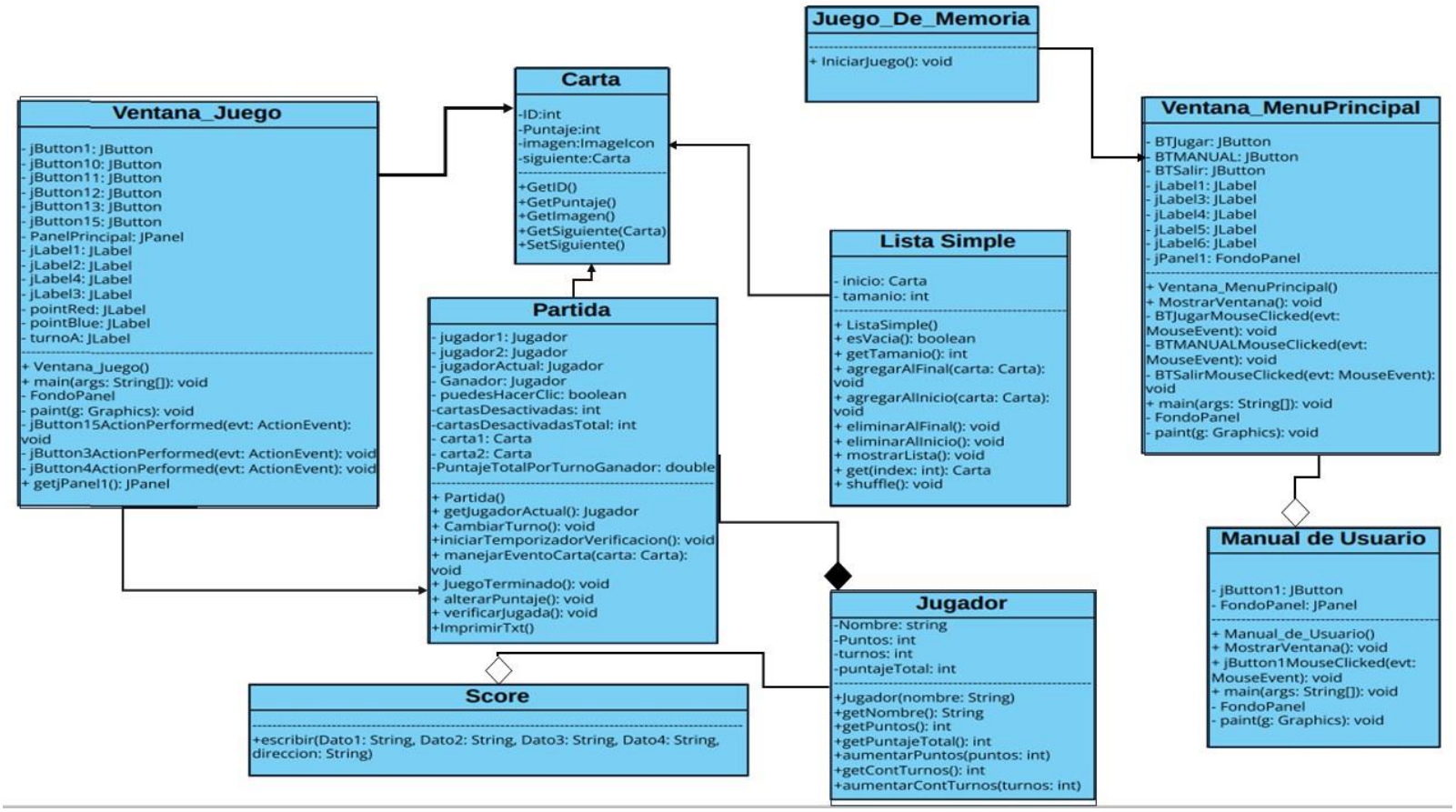
Una de las mayores trabas que se tuvo a la hora de la escritura del código, fue encontrar una manera de voltear las cartas. Esto se soluciono de una forma muy simple, la clase Carta pasó a ser subclase de la Clase predefinida de Java, Button. Esto nos permitió utilizar todas las propiedades de los botones y añadirle de forma eficiente a todos los botones las características de las cartas. Para voltearlas se hizó uso de la propiedad Enabled, la cual funciona con un valor booleano que desactiva el botón y es alterada mediante un evento del tipo Click; cada que se clickea un botón este se desactiva. Y los botones en Java puede visualizar una imagen cuando están activados y otra cuando es lo contrario. Mediante esta lógica se pudo implementar exitosamente la función de voltear las cartas.

Otro problema que surgió a la hora de desarrollar este programa, fue la de barajear las cartas. Se tenia en mente usar métodos de desordenamiento como el Fisher-Yates, método el cual fue adaptado e implementado en la función Shuffle de nuestra clase ListaSimple. Al desordenar la lista, sus elementos son añadidos a un panel de la interfaz gráfica mediante una estructura repetitiva de tipo while. Este panel fue un gran reto debido a que no podíamos previsualizar su diseño, ya que sus elementos eran añadidos luego de la compilación del código.

Finalmente nos decidimos por crear dos ventanas adicionales, las cuales representarían el menú principal y el manual de usuario respectivamente; este proceso sería relativamente rápido puesto que los únicos métodos resaltantes fueron: Un método que coloca fondos a cada Jframe y un método para moverse entre ventanas. Además se añadió la visualización de los datos de cada jugador, actualizando constantemente el contenido de labels en la interfaz grafica.

Una vez terminada la interfaz gráfica del menú principal, del manual de usuario y de la ventana de juego, solo nos quedaba anexar un registro de los ganadores mediante un archivo de texto, el cual registra: El nombre genérico del ganador, la cantidad de puntos parciales que obtuvo, la cantidad de intentos que le tomó ganar y el puntaje total con el que ganó.

A continuación se anexa el diagrama de clases del proyecto:



Análisis de las Clases

Clase Carta:

- Esta clase representa una carta del juego de memoria.

- Atributos:

  - `ID`: un número entero que identifica de forma única la carta.

  - `Puntaje`: un número entero que representa el valor o puntaje de la carta.

  - `imagen`: un objeto ImageIcon que almacena la imagen asociada a la carta.

  - `siguiente`: una referencia a la siguiente carta en la lista de cartas.

- Métodos:

  - El constructor `Carta(ID, Puntaje, imagen)` inicializa los atributos de la carta.

  - Métodos `get` para obtener los atributos de la carta.

  - Métodos `set` para establecer la siguiente carta.

Clase Jugador:

- Esta clase representa a un jugador en el juego de memoria.

- Atributos:

  - `nombre`: una cadena de caracteres que almacena el nombre del jugador.

  - `puntos`: un número entero que guarda la puntuación del jugador.

  - `turnos`: un número entero que cuenta los turnos del jugador.

- Métodos:

  - El constructor `Jugador(nombre)` inicializa los atributos del jugador.

  - Métodos `get` para obtener el nombre, puntos y contador de turnos del jugador.

  - Métodos `aumentarPuntos` y `aumentarContTurnos` para modificar los puntos y el contador de turnos del jugador.

Clase ListaSimple\<X\>:

- Esta clase representa una lista simple genérica que se utiliza para almacenar las cartas.

- Atributos:

  - `Inicio`: una referencia al primer elemento de la lista.

  - `Tamaño`: un número entero que indica el tamaño de la lista.

- Métodos:

  - La clase define métodos para agregar, eliminar, mostrar elementos, obtener el tamaño y mezclar la lista.

Clase Partida:

- Esta clase gestiona una partida del juego de memoria.

- Atributos:

  - Instancias de la clase `Jugador` para representar a los jugadores.

  - `puedesHacerClic`: un booleano que controla si el jugador puede hacer clic en las cartas.

  - `cartasDesactivadas` y `cartasDesactivadasTotal`: números enteros que registran la cantidad de cartas desactivadas y el total de cartas desactivadas.

  - `carta1` y `carta2`: referencias a las cartas seleccionadas por el jugador.

- Métodos:

  - `obtenerJugadorActual` devuelve el jugador actual.

- `CambiarTurno` cambia el turno entre los jugadores.

- `iniciarTemporizadorVerificacion` inicia un temporizador para verificar la jugada.

- `manejarEventoCarta` gestiona el evento de hacer clic en una carta.

- `JuegoTerminado` determina y muestra el ganador de la partida.

- `actualizarPuntaje` actualiza el puntaje del jugador actual.

- `verificarJugada` verifica si las cartas seleccionadas por el jugador forman un par y realiza las acciones correspondientes.

Clase Ventana_Juego:

- Esta clase representa la interfaz gráfica del juego de memoria.

- Configura las cartas y su interacción, así como la lógica del juego.

Clase Manual_de_Usuario:

- Esta clase representa la ventana de manual de usuario.

- Permite a los jugadores acceder a la información sobre cómo jugar el juego.

Clase FondoPanel:

- Esta clase personalizada se utiliza para establecer un fondo de imagen en la ventana de manual de usuario.

Método Main:

- En el método principal, se inicia el juego de memoria y se configuran las cartas, jugadores y la ventana de juego.

Pseudocódigo : Paquete Feactures

Pseudocodigo Carta

```
// Clase Carta
clase Carta
    // Atributos
    ID: entero
    Puntaje: entero
    imagen: ImageIcon
    siguiente: Carta

    // Constructor de la clase
    constructor Carta(ID: entero, Puntaje: entero, imagen: ImageIcon)
        this.ID = ID
        this.Puntaje = Puntaje
        this.imagen = imagen
        this.siguiente = nulo
    Fin

    // Método para obtener el ID de la carta
    método getID(): entero
        retornar ID
    Fin

    // Método para obtener el puntaje de la carta
    método getPuntos(): entero
        retornar Puntaje
    Fin
```

```
    // Método para obtener la imagen de la carta
    método getImagen(): ImageIcon
        retornar imagen
    Fin

    // Método para obtener la siguiente carta
    método getSiguiente(): Carta
        retornar siguiente
    Fin

    // Método para establecer la siguiente carta
    método setSiguiente(siguiente: Carta)
        this.siguiente = siguiente
    Fin
Fin
```

Fin Pseudocodigo Carta

Pseudocodigo Jugador

```
// Clase Jugador
clase Jugador
    // Atributos
    nombre: cadena
    puntos: entero
    turnos: entero
    // Constructor de la clase
    constructor Jugador(nombre: cadena)
        this.nombre = nombre
        this.puntos = 0
```

```
        this.turnos = 0
    Fin

    // Método para obtener el nombre del jugador
    método getNombre(): cadena
        retornar nombre
    Fin

    // Método para obtener los puntos del jugador
    método getPuntos(): entero
        retornar puntos
    Fin

    // Método para aumentar los puntos del jugador
    método aumentarPuntos(puntos: entero)
        this.puntos += puntos
    Fin

    // Método para obtener el contador de turnos del jugador
    método getContTurnos(): entero
        retornar turnos
    Fin
    // Método para aumentar el contador de turnos del jugador
    método aumentarContTurnos(turnos: entero)
        this.turnos += turnos
    Fin
Fin
```

Fin Pseudocodigo Jugador

Pseudocodigo ListaSimple

Clase Carta:

Propiedades:

- ID
- Puntaje
- Imagen
- Siguiente

Clase ListaSimple<X>:

Propiedades:

- Inicio
- Tamaño

Método esVacia():

Devuelve verdadero si el Inicio es nulo, de lo contrario, falso.

Método getTamanio():

Devuelve el valor de la propiedad Tamaño.

Método agregarAlFinal(Carta carta):

Si la lista está vacía:

Establecer Inicio como la carta.

De lo contrario:

Crear una variable auxiliar y asignarla a Inicio.

Mientras la propiedad Siguiente de auxiliar no sea nula:

Mover auxiliar al siguiente elemento en la lista.

Establecer la propiedad Siguiente de auxiliar como la carta.

Incrementar el Tamaño.

Método agregarAlInicio(Carta carta):

Establecer la propiedad Siguiente de carta como Inicio.

Establecer Inicio como la carta.

Incrementar el Tamaño.

Método eliminarAlFinal():

Si la lista no está vacía:

Si la propiedad Siguiente de Inicio es nula:

Establecer Inicio como nulo.

De lo contrario:

Crear una variable auxiliar y asignarla a Inicio.

Mientras la propiedad Siguiente de la propiedad Siguiente de auxiliar no sea nula:

Mover auxiliar al siguiente elemento en la lista.

Establecer la propiedad Siguiente de auxiliar como nula.

Decrementar el Tamaño.

Método eliminarAlInicio():

Si la lista no está vacía:

Establecer Inicio como la propiedad Siguiente de Inicio.

Decrementar el Tamaño.

Método mostrarLista():

Crear una variable auxiliar y asignarla a Inicio.

Mientras auxiliar no sea nulo:

Imprimir "ID: " + ID de auxiliar.

Mover auxiliar al siguiente elemento en la lista.

Imprimir "null".

Método get(int index):

Si el índice es menor que 0 o mayor o igual al Tamaño:

Lanzar una excepción de "Índice fuera de los límites de la lista".

Crear una variable auxiliar y asignarla a Inicio.

Para i desde 0 hasta index - 1:

Mover auxiliar al siguiente elemento en la lista.

Devolver auxiliar.

Método shuffle():

Crear una instancia de la clase Random.

Obtener el valor de Tamaño.

Crear un arreglo de cartas de tamaño Tamaño.

Crear una variable actual y asignarla a Inicio.

Para i desde 0 hasta Tamaño - 1:

Asignar la carta actual al elemento i del arreglo.

Mover actual al siguiente elemento en la lista.

Para i desde Tamaño - 1 hasta 1:

Generar un número aleatorio j entre 0 y i (inclusive).

Intercambiar las cartas en las posiciones i y j del arreglo.

Fin Pseudocodigo ListaSimple

Pseudocodigo Partida

```
Clase Partida:
    Propiedades:
        Jugador jugador1
        Jugador jugador2
        Jugador jugadorActual
        Jugador Ganador
        Boolean puedesHacerClic
        Entero cartasDesactivadas
        Entero cartasDesactivadasTotal
        Carta carta1
        Carta carta2

    Constructor Partida():
        jugador1 = Nuevo Jugador("Red")
        jugador2 = Nuevo Jugador("Blue")
        jugadorActual = jugador1
        jugadorActual.aumentarContTurnos(1)

    Método obtenerJugadorActual():
        Devuelve jugadorActual

    Método CambiarTurno():
        Si jugadorActual es igual a jugador1:
            jugadorActual = jugador2
            ActualizaInterfaz("Blue")
        De lo contrario:
            jugadorActual = jugador1
```

ActualizaInterfaz("Red")

Método iniciarTemporizadorVerificacion():

Crear Temporizador con retraso de 800 ms y ActionListener que llama verificarJugada

Configurar el Temporizador para no repetir

Iniciar el Temporizador

Método manejarEventoCarta(Carta carta):

Si puedesHacerClic es verdadero y carta está habilitada:

Desactivar carta

Incrementar cartasDesactivadas en 1

Si cartasDesactivadas es 1:

carta1 = carta

De lo contrario, si cartasDesactivadas es 2:

carta2 = carta

puedesHacerClic = falso

Llamar iniciarTemporizadorVerificacion

Método JuegoTerminado():

Calcular PuntajePorTurnoJugador1 = (puntos de jugador1 / turnos de jugador1)

Calcular PuntajePorTurnoJugador2 = (puntos de jugador2 / turnos de jugador2)

Si PuntajePorTurnoJugador1 > PuntajePorTurnoJugador2:

Ganador = jugador1

De lo contrario:

Ganador = jugador2

MostrarMensaje("¡Felicidades, has ganado " + Ganador.nombre + "!", "Juego Terminado")

// Reemplazar esto con el manejo real del final del juego

Método actualizarPuntaje():

puntajeTexto = ConvertirAString(jugadorActual.puntos)

Si jugadorActual es igual a jugador1:

ActualizarInterfazPuntaje("Red", puntajeTexto)

De lo contrario:

ActualizarInterfazPuntaje("Blue", puntajeTexto)

Método verificarJugada():

Si carta1 y carta2 no son nulos:

MostrarMensaje("La cantidad usada de turnos de " + jugadorActual.nombre + " es: " + jugadorActual.turnos)

Si carta1.id es igual a carta2.id:

Establecer carta1 y carta2 como nulos

Aumentar el puntaje del jugador actual en 1

Llamar a actualizarPuntaje()

Incrementar cartasDesactivadasTotal en 1

MostrarMensaje("EL puntaje de " + jugadorActual.nombre + " es: " + jugadorActual.puntos + " puntos")

De lo contrario:

Habilitar carta1 y carta2

Llamar a CambiarTurno()

MostrarMensaje("Turno de " + jugadorActual.nombre + "!", "Cambio de Turno")

Incrementar los turnos del jugador actual en 1

Si cartasDesactivadasTotal es igual a 9:

Llamar a JuegoTerminado()

Establecer puedesHacerClic como verdadero

Establecer cartasDesactivadas en 0

Fin Pseudocodigo Partida

Pseudocódigo : Paquete Juego_Memoria

Pseudocodigo Juego_Memoria

```
inicio
    // Declaración de clases y variables
    Ventana_Juego: JFrame
    Partida: Partida
    MazoCartas: ListaSimple<Carta>
    imagen: ImageIcon
    rolloverIcon: ImageIcon

    // Inicializar la ventana del juego
    ventana = nueva Ventana_Juego()
    partida = nueva Partida()
    imagen = nueva ImageIcon(Carta.class.getResource("/Imagenes/Carta0.jpg"))
    rolloverIcon = nueva ImageIcon(Carta.class.getResource("/Imagenes/CartaR.jpg"))

    // Crear y configurar las cartas
    para i de 1 hasta 9
        para j de 1 hasta 2
            nuevaCarta = nueva Carta(i, 5, imagen)
            Fondodecarta = nueva ImageIcon(Carta.class.getResource("/Imagenes/Carta" +
i + ".jpg"))
            MazoCartas.agregarAlFinal(nuevaCarta)
            nuevaCarta.setIcon(imagen)
            nuevaCarta.setRolloverIcon(rolloverIcon)
            nuevaCarta.setDisabledIcon(Fondodecarta)
            nuevaCarta.setBorderPainted(falso)
            nuevaCarta.setMargin(nuevo Insets(0, 0, 0, 0))
            nuevaCarta.setSize(150, 221)
```

```
            // Interacción con las cartas
            nuevaCarta.addActionListener(nueva ActionListener() {
                método actionPerformed(e: ActionEvent)
                    partida.manejarEventoCarta(nuevaCarta)
                Fin
            })

            // Fin de la interacción con las cartas
        Fin
    Fin

    // Mezclar las cartas
    MazoCartas.shuffle()

    // Agregar las cartas al panel en el nuevo orden aleatorio
    actual = MazoCartas.get(0)
    mientras actual no sea nulo
        ventana.getjPanel1().add(actual)
        actual = actual.getSiguiente()
    Fin

    // Mostrar la ventana del juego
    ventana.setVisible(verdadero)
Fin
```

Fin Pseudocodigo Juego_Memoria

Pseudocodigo Manual_de_Usuario

```
inicio
    // Declaración de clases y variables
    FondoPanel: JPanel
    imagen: Image

    // Inicializar la ventana de Manual de Usuario
    ventana = nueva Manual_de_Usuario()

    // Método para mostrar la ventana
    método MostrarVentana()
        nueva Manual_de_Usuario().setVisible(verdadero)
    Fin

    // Creación y configuración de la ventana
    ventana.setContentPane(nueva FondoPanel())
    ventana.initComponents()
Fin

// Clase Manual_de_Usuario
clase Manual_de_Usuario
    // Constructor de la clase
    constructor Manual_de_Usuario()
        FondoPanel fondoPanel = nueva FondoPanel()
        setContentPane(fondoPanel)
        initComponents()
    Fin
```

```
    // Método para mostrar la ventana
    método MostrarVentana()
        nueva Manual_de_Usuario().setVisible(verdadero)
    Fin

    // Inicio de la definición de componentes
    método initComponents()
        jButton1 = nueva JButton()

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)

        jButton1.setIcon(nueva ImageIcon(getClass().getResource("/Imagenes/Boton para
Regresar.png")))
        jButton1.setBorder(BorderFactory.createLineBorder(nueva Color(0, 0, 0)))
        jButton1.setRolloverIcon(nueva
ImageIcon(getClass().getResource("/Imagenes/Boton para Regresar
Seleccionado.png")))
        jButton1.addMouseListener(nueva MouseAdapter() {
            método mouseClicked(evt: MouseEvent)
                Ventana_MenuPrincipal.MostrarVentana()
                this.dispose()
            Fin
        })

        // Configuración del layout
        layout = nueva GroupLayout(getContentPane())
        getContentPane().setLayout(layout)
        layout.setHorizontalGroup(
            layout.createParallelGroup(GroupLayout.Alignment.TRAILING)
```

```
            .addGroup(GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(804, Short.MAX_VALUE)
                .addComponent(jButton1)
                .addGap(98, 98, 98)
            )
        )
        layout.setVerticalGroup(
            layout.createParallelGroup(GroupLayout.Alignment.TRAILING)
            .addGroup(GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(544, Short.MAX_VALUE)
                .addComponent(jButton1)
                .addContainerGap()
            )
        )

        pack()
    Fin
    // Fin de la definición de componentes

    // Inicio de la definición de eventos
    método jButton1MouseClicked(evt: MouseEvent)
        Ventana_MenuPrincipal.MostrarVentana()
        this.dispose()
    Fin
    // Fin de la definición de eventos
Fin
```

```
// Clase FondoPanel
clase FondoPanel
    // Declaración de variables
    imagen: Image

    // Método paint
    método paint(g: Graphics)
        imagen = nueva ImageIcon(getClass().getResource("/Imagenes/Manual de Usuario Final.jpg")).getImage()
        g.drawImage(imagen, 0, 0, getWidth(), getHeight(), this)
        setOpaque(falso)
        super.paint(g)
    Fin
Fin
```

Fin Pseudocodigo Manual_de_Usuario

Pseudocodigo Ventana_MenuPrincipal

```
inicio
    // Declaración e inicialización de componentes
    BTJugar: JButton
    BTMANUAL: JButton
    BTSalir: JButton

    // Crear ventana de menú principal
    Ventana_MenuPrincipal: JFrame
    Ventana_MenuPrincipal.setLayout(new BorderLayout())
    Ventana_MenuPrincipal.setDefaultCloseOperation(EXIT_ON_CLOSE)
    Ventana_MenuPrincipal.setResizable(falso)
    Ventana_MenuPrincipal.setSize(anchura: 800, altura: 600)

    // Panel de fondo
    FondoPanel: JPanel
    FondoPanel.setLayout(null)
    FondoPanel.setBackground(color: blanco)

    // Botón "Jugar"
    BTJugar = nuevo JButton()
    BTJugar.setIcon(Imagen("/Imagenes/Boton Para Jugar.jpg"))
    BTJugar.setBorder(LíneaNegra(grosor: 2))
    BTJugar.setRolloverIcon(Imagen("/Imagenes/Boton Para Jugar Seleccionado.jpg"))
    BTJugar.addMouseListener(Hacer clic en el ratón(evt))
    BTJugarMouseClicked(evt)
        // Acción al hacer clic en "Jugar"
        Cerrar(Ventana_MenuPrincipal)
```

```
        IniciarJuego()
    Fin

    // Botón "Manual"
    BTMANUAL = nuevo JButton()
    BTMANUAL.setIcon(Imagen("/Imagenes/Boton Para Manual de Usuario.jpg"))
    BTMANUAL.setBorder(LíneaNegra(grosor: 2))
    BTMANUAL.setRolloverIcon(Imagen("/Imagenes/Boton Para Manual de Usuario
Seleccionado.jpg"))
    BTMANUAL.addMouseListener(Hacer clic en el ratón(evt))
    BTMANUALMouseClicked(evt)
        // Acción al hacer clic en "Manual"
        Cerrar(Ventana_MenuPrincipal)
        MostrarManualDeUsuario()
    Fin

    // Botón "Salir"
    BTSalir = nuevo JButton()
    BTSalir.setIcon(Imagen("/Imagenes/Boton para Salir.jpg"))
    BTSalir.setBorder(LíneaNegra(grosor: 2))
    BTSalir.setRolloverIcon(Imagen("/Imagenes/Boton para Salir Seleccionado.jpg"))
    BTSalir.addMouseListener(Hacer clic en el ratón(evt))
    BTSalirMouseClicked(evt)
        // Acción al hacer clic en "Salir"
        Cerrar(Ventana_MenuPrincipal)
    Fin

    // Etiqueta de autoría
    EtiquetaAutoria: JLabel
```

```
    EtiquetaAutoria.setText("Made by: MC and GV")
    EtiquetaAutoria.setFont(Fuente("Dialog", negrita: verdadero, tamaño: 18))
    EtiquetaAutoria.setForeground(color: negro)

    // Agregar componentes al panel de fondo
    FondoPanel.AgregarComponente(BTJugar)
    FondoPanel.AgregarComponente(BTMANUAL)
    FondoPanel.AgregarComponente(BTSalir)
    FondoPanel.AgregarComponente(EtiquetaAutoria)

    // Posicionamiento de componentes en el panel de fondo
    BTJugar.setPosición(x: 330, y: 100)
    BTMANUAL.setPosición(x: 330, y: 200)
    BTSalir.setPosición(x: 330, y: 300)
    EtiquetaAutoria.setPosición(x: 330, y: 450)

    // Agregar panel de fondo a la ventana
    Ventana_MenuPrincipal.Agregar(FondoPanel)

    // Mostrar la ventana de menú principal
    Ventana_MenuPrincipal.Mostrar()
Fin
```

Fin Pseudocodigo Ventana_MenuPrincipal

Pseudocodigo Ventana_Juego

```
inicio
    // Declaración de clases y variables
    Ventana_Juego: JFrame
    FondoPanel: Clase
    jButton1: JButton
    jButton10: JButton
    jButton11: JButton
    jButton12: JButton
    jButton13: JButton
    jButton15: JButton
    PanelPrincipal: JPanel
    jLabel1: JLabel
    jLabel2: JLabel
    jLabel3: JLabel
    jLabel4: JLabel
    pointRed: JLabel
    pointBlue: JLabel
    turnoA: JLabel
    turnoB: JLabel
    jButton2: JButton
    jButton3: JButton
    jButton4: JButton

    // Inicialización de la ventana
    Ventana_Juego = nueva Ventana_Juego()
    Ventana_Juego.setContentPane(FondoPanel())
    MusicPlayer: player = nueva MusicPlayer()
```

```
    Ventana_Juego.inicializarComponentes()

    // Definición de métodos
    método jButton15ActionPerformed(evt)
        // Acción al hacer clic en jButton15
    Fin

    método jButton3ActionPerformed(evt)
        // Acción al hacer clic en jButton3
        Cerrar(Ventana_Juego)
        Ventana_MenuPrincipal.MostrarVentana()
    Fin

    método jButton4ActionPerformed(evt)
        // Acción al hacer clic en jButton4
        Cerrar(Ventana_Juego)
    Fin

    Fin
Fin
```

Fin Pseudocodigo Ventana_Juego