



MINIPROYECTO 3: BATALLA NAVAL

ANÁLISIS DE CLASES

Barco

- ❖ Responsabilidades
 - 1. Definir y retornar las características de los barcos.
 - 2. Asignar la posición de los barcos al tablero.
- ❖ Funciones
 - barcosPosicionados()

```
public boolean barcosPosicionados() {  
    if (contadorPortaaviones == 4) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

- getContadorFragatas()
- getContadorDestructores()
- getContadorSubmarinos()
- getContadorPortaaviones()

```
public static int getContadorFragatas() { return contadorFragatas; }  
  
public static int getContadorDestructores() { return contadorDestructores; }  
  
public static int getContadorSubmarinos() { return contadorSubmarinos; }  
  
public static int getContadorPortaaviones() { return contadorPortaaviones; }
```

➤ `asignarPosicionFragatas(int x, int y)`

```
public void asignarPosicionFragatas(int x, int y) {
    array[0] = x;
    array[1] = y;
    contadorFragatas = contadorFragatas + 1;
    System.out.println("Fragatas");
    for (int i = 0; i < fragatasx.length; i++) {
        if (fragatasx[i] == 0) {
            fragatasx[i] = array[0];
            fragatasy[i] = array[1];
            //System.out.println(fragatasx[i] + ", " + fragatasy[i]);
            i = fragatasx.length + 1;
        }
    }
}
```

➤ `asignarPosicionesDestructores(int x, int y)`

```
public void asignarPosicionesDestructores(int x, int y) {
    array[0] = x;
    array[1] = y;
    contadorDestructores = contadorDestructores + 1;
    System.out.println("Destructores");
    for (int i = 0; i < destructoresx.length; i++) {
        if (destructoresx[i] == 0) {
            destructoresx[i] = array[0];
            destructoresy[i] = array[1];
            //System.out.println(destructoresx[i] + ", " + destructoresy[i]);
            i = destructoresx.length + 1;
        }
    }
}
```

➤ `asignarPosicionesSubmarinos(int x, int y)`

```
public void asignarPosicionesSubmarinos(int x, int y) {
    array[0] = x;
    array[1] = y;
    contadorSubmarinos = contadorSubmarinos + 1;
    System.out.println("Submarinos");
    for (int i = 0; i < destructoresx.length; i++) {
        if (submarinosx[i] == 0) {
            submarinosx[i] = array[0];
            submarinosy[i] = array[1];
            //System.out.println(submarinosx[i] + ", " + submarinosy[i]);
            i = submarinosx.length + 1;
        }
    }
}
```

➤ asignarPosicionesPortaaviones(int x, int y)

```
public void asignarPosicionesPortaaviones(int x, int y) {  
    array[0] = x;  
    array[1] = y;  
    contadorPortaaviones = contadorPortaaviones + 1;  
    System.out.println("Portaaviones");  
    for (int i = 0; i < destructoresx.length; i++) {  
        if (portaAvionesx[i] == 0) {  
            portaAvionesx[i] = array[0];  
            portaAvionesy[i] = array[1];  
            //System.out.println(portaAvionesx[i] + ", " + portaAvionesy[i]);  
            i = portaAvionesx.length + 1;  
        }  
    }  
}
```

TableroCPU

- ❖ Responsabilidades
 - 1. Generar el tablero para el enemigo, en este caso CPU.
- ❖ Funciones
 - tableroCPU()

```
public class tableroCPU extends JPanel {  
  
    public tableroCPU() {  
  
        JLabel[][] matrizLabelCPU = new JLabel[11][11];  
  
        this.setPreferredSize(new Dimension(width:390, height:390));  
        this.setBorder(BorderFactory.createTitledBorder("Tablero CPU"));  
        this.setLayout(null);  
  
        for (int i = 0; i < 11; i++) {  
            for (int j = 0; j < 11; j++) {  
                matrizLabelCPU[i][j] = new JLabel(text:"", SwingConstants.CENTER);  
                matrizLabelCPU[i][j].setBorder(new LineBorder(Color.BLACK));  
                matrizLabelCPU[i][j].setOpaque(true);  
                matrizLabelCPU[i][j].setBounds(x:30 + j * 30, y:30 + i * 30, width:30, height:30);  
                matrizLabelCPU[i][j].setBackground(Color.cyan);  
                matrizLabelCPU[i][j].setPreferredSize(new Dimension(width:30, height:30));  
                if (i == 0 || j == 0) {}  
                else {  
                    matrizLabelCPU[i][j].addMouseListener(GUIGridBagLayout.escucha);  
                }  
                this.add(matrizLabelCPU[i][j]);  
            }  
        }  
    }  
}
```

- generarFragatas()
- generarDestructores()
- generarPortaaviones()
- generarSubmarinos()

```
public static void generarFragatas(){

}

public static void generarDestructores(){

}

public static void generarSubmarinos(){

}

public static void generarPortaaviones(){

}
```

- tieneVecinosDisponibles(int i, int j)

```
public boolean tieneVecinosDisponibles(int i, int j) {
    boolean valor = false;
    switch (posicionando) {
        case 0:
            valor = true;
            break;
        case 1:
            if (verificarArriba(i, j, 1) == true || verificarAbajo(i, j, 1) == true || verificarDe
                valor = true;
            } else {
                valor = false;
            }
            break;
        case 2:
            if (parteColocandoSubmarino == 0) {
                if ((verificarArriba(i, j, 1) == true && verificarArriba(i, j, 2) == true) || (ver
                    valor = true;
                } else {
                    if ((verificarDerecha(i, j, 1) == true && verificarDerecha(i, j, 2) == true) | |
                        valor = true;
                    } else {
                        if (verificarArriba(i, j, 1) == true && verificarAbajo(i, j, 1) == true) {
                            valor = true;
                        } else if (verificarDerecha(i, j, 1) == true && verificarIzquierda(i, j, 1)
                            valor = true;
                        }
                    }
                }
            }
        }
    }
}
```

- sePuedePosicionar(int i, int j)

```
public boolean sePuedePosicionar(int i, int j) {
    if (matrizLabelCPU[i][j].getBackground() == Color.cyan && tieneVecinosDisponibles(i, j) == tr
        if (posicionPresionadaDisponible(i, j) == true) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

➤ `posicionPresionadaDisponible(int i, int j)`

```
public boolean posicionPresionadaDisponible(int i, int j) {  
  
    boolean valor = true;  
    Vector<Integer> posibleArriba = new Vector<~>();  
    Vector<Integer> posibleAbajo = new Vector<~>();  
    Vector<Integer> posibleDerecha = new Vector<~>();  
    Vector<Integer> posibleIzquierda = new Vector<~>();  
  
    switch (posicionando) {  
        case 0:  
            valor = true;  
            break;  
        case 1:  
            if (parteColocandoDestruidores == 0) {  
                primerox = i;  
                primeroy = j;  
                valor = true;  
                if (primerox + 1 < 11) {  
                    posibleArriba.add(primerox + 1);  
                    posibleArriba.add(primeroy);  
                    posicionesPosibles.add(posibleArriba);  
                }  
                if (primerox - 1 > 0) {  
                    posibleAbajo.add(primerox - 1);  
                }  
            }  
    }  
}
```

GUIGridBagLayout

❖ Responsabilidades

1. Grafica cada una de las fases del juego.
2. Grafica tanto el tablero de la CPU como el del jugador.
3. Darle funcionalidad a los escuchas del juego.
4. Grafica la posición de los barcos.

❖ Funciones

➤ `initGridBagLayout()`

```

private void initGUI() {
    //Set up JFrame Container's Layout
    this.setLayout(new GridLayout());
    GridBagConstraints constraints = new GridBagConstraints();
    //Create Listener Object or Control Object
    escucha = new Escucha();
    mimodeloBatallaNaval = new ModelBatallaNaval();
    miBarcoGui = new Barco();
    miTableroCPU = new tableroCPU();
    miTableroPersonal = new tableroPersonal();
    //Set up JComponents

    headerProject = new Header(title:"Batalla Naval", Color.BLACK);
    constraints.gridx = 0;
    constraints.gridy = 0;
    constraints.gridwidth = 2;
    constraints.fill = GridBagConstraints.HORIZONTAL;
    add(headerProject, constraints);

    {
        constraints.gridx = 0;
        constraints.gridy = 1;
    }
}

```

ModelBatallaNaval

❖ Responsabilidades

1. Dar valores iniciales a todos los atributos.
2. Determinar las zonas disponibles para posicionar un barco.
- 3.

❖ Funciones

- calcularEstado()

```

public void calcularEstado() {
    if (miBarco.barcosPosicionados() == false) {
        estado = 0;
    } else if (miBarco.barcosPosicionados() == true) {
        estado = 1;
    }
}

```

- sePuedePosicionar(int i, int j)

```

public boolean sePuedePosicionar(int i, int j) {
    if (tableroPersonal.matrizLabel[i][j].getBackground() == Color.cyan && tieneVecinosDisponibles(i,
        if (posicionPresionadaDisponible(i, j) == true) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}

```

➤ tieneVecinosDisponibles(int i, int j)

```
public boolean tieneVecinosDisponibles(int i, int j) {  
    boolean valor = false;  
    switch (posicionando) {  
        case 0:  
            valor = true;  
            break;  
        case 1:  
            if (verificarArriba(i, j, numero:1) == true || verificarAbajo(i, j, numero:1) == true || verificarDerecha(i, j, numero:1) == true)  
                valor = true;  
            } else {  
                valor = false;  
            }  
            break;  
        case 2:  
            if (parteColocandoSubmarino == 0) {  
                if ((verificarArriba(i, j, numero:1) == true && verificarArriba(i, j, numero:2) == true)  
                    valor = true;  
                } else {  
                    if ((verificarDerecha(i, j, numero:1) == true && verificarDerecha(i, j, numero:2) == true)  
                        valor = true;  
                    } else {  
                        if (verificarArriba(i, j, numero:1) == true && verificarAbajo(i, j, numero:1) == true)  
                            valor = true;  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

➤ verificarArriba(int i, int j, int numero)

```
public boolean verificarArriba(int i, int j, int numero) {  
    if (i + numero > 10) {  
        return false;  
    } else {  
        if (tableroPersonal.matrizLabel[i + numero][j].getBackground() == Color.cyan) {  
            return true;  
        } else if (tableroPersonal.matrizLabel[i + numero][j].getBackground() == colorPoniendo) {  
            System.out.println("Tiene disponible arriba");  
            return true;  
        } else {  
            System.out.println("No tiene disponible arriba");  
            return false;  
        }  
    }  
}
```

➤ VerificarAbajo(int i, int j, int numero)

```
public boolean verificarAbajo(int i, int j, int numero) {  
    if (i - numero < 1) {  
        return false;  
    } else {  
        if (tableroPersonal.matrizLabel[i - numero][j].getBackground() == Color.cyan) {  
            return true;  
        } else if (tableroPersonal.matrizLabel[i - numero][j].getBackground() == colorPoniendo) {  
            System.out.println("Tiene disponible abajo");  
            return true;  
        } else {  
            System.out.println("No tiene disponible abajo");  
            return false;  
        }  
    }  
}
```

➤ verificarDerecha(int i, int j, int numero)

```
public boolean verificarDerecha(int i, int j, int numero) {  
    if (j + numero > 10) {  
        return false;  
    } else {  
        if (tableroPersonal.matrizLabel[i][j + numero].getBackground() == Color.cyan) {  
            return true;  
        } else if (tableroPersonal.matrizLabel[i][j + numero].getBackground() == colorPoniendo) {  
            System.out.println("Tiene disponible derecha");  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

➤ verificarIzquierda(int i, int j, int numero)

```
public boolean verificarIzquierda(int i, int j, int numero) {  
    if (j - numero < 1) {  
        return false;  
    } else {  
        if (tableroPersonal.matrizLabel[i][j - numero].getBackground() == Color.cyan) {  
            return true;  
        } else if (tableroPersonal.matrizLabel[i][j - numero].getBackground() == colorPoniendo) {  
            System.out.println("Tiene disponible izquierda");  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

➤ posicionDisponible(int i, int j)

```
public boolean posicionPresionadaDisponible(int i, int j) {  
  
    boolean valor = true;  
    Vector<Integer> posibleArriba = new Vector<~>();  
    Vector<Integer> posibleAbajo = new Vector<~>();  
    Vector<Integer> posibleDerecha = new Vector<~>();  
    Vector<Integer> posibleIzquierda = new Vector<~>();  
  
    switch (posicionando) {  
        case 0:  
            valor = true;  
            break;  
        case 1:  
            if (parteColocandoDestruidores == 0) {  
                primerox = i;  
                primeroy = j;  
                valor = true;  
                if (primerox + 1 < 11) {  
                    posibleArriba.add(primerox + 1);  
                    posibleArriba.add(primeroy);  
                    posicionesPosibles.add(posibleArriba);  
                }  
                if (primerox - 1 > 0) {  
                    posibleAbajo.add(primerox - 1);  
                    posibleAbajo.add(primeroy);  
                    posicionesPosibles.add(posibleAbajo);  
                }  
            }  
    }  
}
```

tableroPersonal

- ❖ Responsabilidades
 - 1. Generar el tablero para el jugador.
- ❖ Funciones
 - tableroPersonal()

```
public tableroPersonal() {  
  
    this.setLayout(null);  
    this.setPreferredSize(new Dimension(width:390, height:390));  
    this.setSize(width:390, height:390);  
    this.setBorder(BorderFactory.createTitledBorder("Tu tablero"));  
  
    for (int i = 0; i < 11; i++) {  
        for (int j = 0; j < 11; j++) {  
            matrizLabel[i][j] = new JLabel(text:"", SwingConstants.CENTER);  
            matrizLabel[i][j].setBorder(new LineBorder(Color.BLACK));  
            matrizLabel[i][j].setOpaque(true);  
            matrizLabel[i][j].setBounds(x:30 + j * 30, y:30 + i * 30, width:30, height:30);  
            matrizLabel[i][j].setBackground(Color.cyan);  
            matrizLabel[i][j].setPreferredSize(new Dimension(width:30, height:30));  
            if (i == 0 || j == 0) {  
            } else {  
                matrizLabel[i][j].addMouseListener(GUIGridBagLayout.escucha);  
            }  
            this.add(matrizLabel[i][j]);  
        }  
    }  
}
```

BOCETO DE INTERFAZ

