Docuementación de Proyecto: Buscaminas

Nombre: Alejandro José Castellanos Zavala
Número de cuenta: 12441410
Asignatura: 892 Programación I
Catedrático: Ing. Darío Alexander Cardona Aguilar

Fecha: domingo 15 de septiembre de 2024

Introducción

Este proyecto consistió en desarrollar el clásico juego Buscaminas en Java. el juego cuenta con una interfaz gráfica que permite al usuario hacer click sobre los botones para

El juego consiste en despejar todas las celdas que no contengan una mina de entre una matriz de celdas.

Algunas celdas tienen un número, el cual indica la cantidad de minas que hay en las celdas circundantes. Por ejmplo, si una celda tiene el número 3, significa que de las ocho celdas que hay alrededor, tres tienen minas y cinco no. Si se descubre una celda sin número, indica que ninguna de las celdas vecinas contiene minas, y éstas se descubren automáticamente.

Si se descubre una celda con una mina, se pierde la partida.

Es posible colocar una marca en las celdas que el jugador cree que conteine minas para ayudar a descubrir las que están cerca. Generalmente esta marca es una bandera.

Elementos del Buscaminas

La implementación del buscaminas creada para el proyecto, consitió principalmente en dos elementos: la matriz secreta y la matriz mostrada.

Matriz Secreta

La matriz secreta contiene la posición de todas las minas y el número de minas circundantes para el resto de las celdas. La matriz secreta es utilizada para comparar y revelar los números del tablero cuando el usuario presiona los botones de la interfaz gráfica.

Colocar Minas

Para inicializar la matriz secreta, primero se colocan las minas aleatoriamente.

```
public void inicializar() {
    limpiarTablero(); // Borrar cualquier mina presente
    for (int i = 0; i < minas; i++) { // Generar minas
        colocarMina();
    }
}
```

Donde el método colocarMinas() es un método recursivo que se llama a sí mismo si la celda elegida aleatoriamente ya contiene una mina.

```
private void colocarMina() {
    int x = random.nextInt(filas);
    int y = random.nextInt(columnas);
    if (matriz[x][y] != MINA) {
        matriz[x][y] = MINA;
```

```
            return;
        }
        colocarMina();
    }
```

Esto asegura de que se coloque exáctamente la cantidad de minas ingresada. Nótese que se podría haber realizado con ciclo while, pero para efectos de aprendizaje, se decidió hacerlo como se indicó anteriormente.

Generar Números

Una vez las minas han sido colocadas, se procede a generar los números para el resto de las celdas. Para esto, recorren todos los elementos de la matriz y para cada uno se hace lo siguiente:

1. Si la celda contiene una mina, saltarla.
2. Se cera una variable cuenta y se inicializa en 0.
3. Se revisan las ocho celdas circundantes con la ayuda de dos ciclos for que van desde [-1, 1].

```
cuenta = 0;
for (int di = -1; di <= 1; di++) {
    for (int dj = -1; dj <= 1; dj++) {
        // Saltar el centro de la celda
        if (di == 0 && dj == 0) {
            continue;
        }
        int newFila = i + di;
        int newCol = j + dj;
        if (esValido(newFila, newCol)) {
            if (matriz[newFila][newCol] == MINA) {
                cuenta++;
            }
        }
    }
```

```
}
matriz[i][j] = (char) (cuenta + CERO);
```

En el código anterior, newFila y newColumna son las coordenadas de las ocho esquinas de una determinada celda.

Para evitar el error OutOfBoundException, se revisa si las coordenadas de las celdas circundantes están dentro de los límites de la matriz, antes de revisar lo que contiene.

```
private boolean esValido(int fila, int columna) {
    return fila >= 0 && fila < filas && columna >= 0 && columna <
columnas;
}
```

A continuación, en la Figura 1 se muestra un diagrama que muestra el proceso mencionado anteriormente.

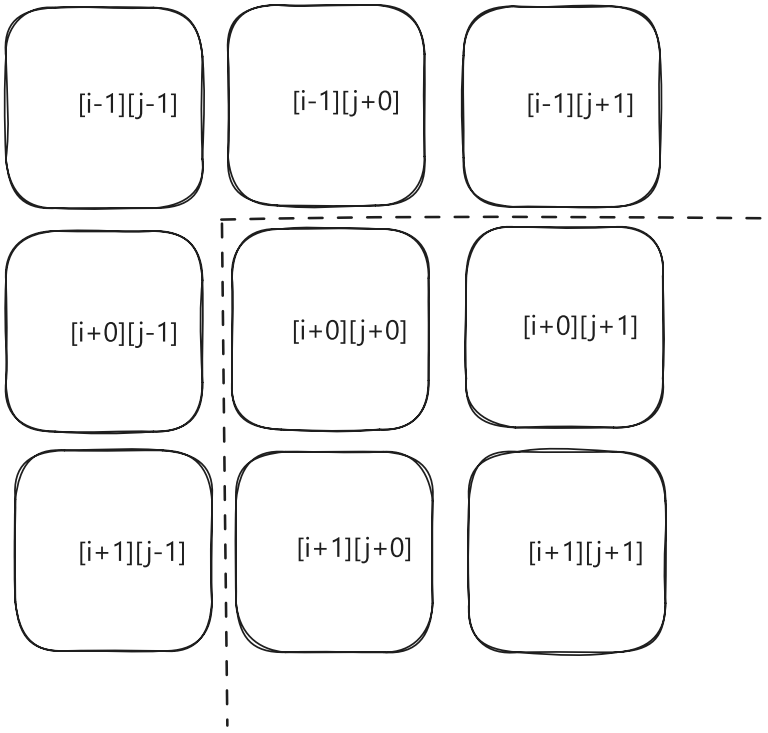

Figura 1: Diagrama de las celdas circundantes para una determinada celda.

Matriz de Botones

Cuando el usuario presiona uno de los botones de la matriz de la interfaz gráfica, se descubre el número que contiene la matriz secreta en esa posición.

Descubrir Celdas

El método para decubrir las celdas se describe a continuación:

1. Si la celda ya ha sido revelada, no hacer nada.
2. Si la celda tiene una bandera, no hacer nada.
3. Si la celda tiene una mina, el juego termina y el jugador pierde.
4. De lo contrario, mostrar el número de la celda.
5. Si la celda contenía un cero, llamar al método descubrirCero para descubrir las celdas circundantes.

```
public void descubrir(int x, int y) {
    if (!buttons[x][y].getIcon().equals(TileIcon)) {
        return; // Saltar si la celda ya ha sido descubierta
    }
    if (buttons[x][y].getIcon().equals(FlagIcon)) {
        return; // Saltar si la celda tiene una bandera
    }
    if (matrizSecreta[x][y] == MINA) {
        juegoPerdido();
        buttons[x][y].setIcon(MinaRojaIcon);
        return;
    }
    // Revelar celda
    buttons[x][y].setIcon(numerosIcons[Character.
    getNumericValue(matrizSecreta[x][y])]);

    if (matrizSecreta[x][y] == CERO) {
        descubrirCeros(x, y);
```

```
    }
}
```

El método para descubrir ceros es muy similar al de generar números mostrado en la Sección 2.1.2. Para cada celda que contenía un cero, se revisan las ocho celdas circundantes y se llama el método descubrir() para revelarlas. Si la celda revelada también contiene un cero, el proceso se repite.

```
private void descubrirCeros(int fila, int columna) {
    for (int di = -1; di <= 1; di++) {
        for (int dj = -1; dj <= 1; dj++) {
            // Saltar el centro de la celda
            if (di == 0 && dj == 0) {
                continue;
            }
            int newFila = fila + di;
            int newCol = columna + dj;

            if (esValido(newFila, newCol) && buttons[newFila]
[newCol].getIcon().equals(TileIcon)) {
                descubrir(newFila, newCol);
            }
        }
    }
}
```

De esta manera se crea un ciclo recursivo, ya que descubirCeros() llama a descubrir() y descubrir() puede llamar a decubrirCeros().

Otros Elementos

Para hacer el diseño más amigable y entretenido, todos los botones tienen figuras, las cuales fueron realizadas por el usuario Tomthedeviant2 (link de DeviantArt).

Por otro lado, se agregó un contador de minas, que se actualiza cada vez que el usuario coloca una bandera, y un temporizador, el cual comienza desde el momento en que el usuario presiona el primer botón y se detiene en el momento en que gana o pierde.

Para esto se hizo uso del objeto Timer que viene incluido dentro de la librería Swing. El objeto toma dos parámetros, el evento y el intervalo entre eventos en milisegundos.

```
private void startTimer() {
    timer = new Timer(1000, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            tiempo++;
            tiempoLabel.setText("Tiempo: " + tiempo);
        }
    });
    timer.start();
}
```

Terminar Juego

El juego termina si el usuario presiona una celda con mina, o si descubre todas las celdas que no contienen minas.

Juego Perdido

Si el jugador pierde, siguiendo el diseño del buscaminas clásico, se muestra en rojo la mina que hizo perder al jugador (la mina que ha estallado) y se muestra la posición del resto. Asimismo, se le indican al usuario todas las suposiciones erroneas que cometió, es decir todas las banderas que colocó sobre celdas que no contenían minas.

```
public void juegoPerdido() {
    tableroBloqueado = true;
    timer.stop();
    faceButton.setIcon(MuertoIcon);

    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            if (matrizSecreta[i][j] == MINA) {
                if (!buttons[i][j].getIcon().equals(FlagIcon)) {
                    buttons[i][j].setIcon(MinaIcon);
                }
            } else if (buttons[i][j].getIcon().equals(FlagIcon))
{
                buttons[i][j].setIcon(MinaXIcon);
            }
        }
    }
}
```

Juego Ganado

Si la suma de todas las celdas en las que el jugador ha puesto una bandera, mas la suma de todas las celdas que no han sido reveladas todavía, es igual a la cantidad de minas, el jugador ganó.

```
public boolean isGanador() {
    int banderas = 0;
    int tiles = 0;
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            if (buttons[i][j].getIcon().equals(FlagIcon)) {
                banderas++;
            }
            if (buttons[i][j].getIcon().equals(TileIcon)) {
                tiles++;
            }
        }
    }

    return banderas + tiles == cantMinas;
}
```

Esto funciona incluso si el jugador ha colocado banderas en celdas que no contienen minas. En ese caso, la suma de las banderas y las celdas no reveladas tendría que ser mayor al número de minas, ya que sería igual a decir que tiene más banderas que minas.

Reiniciar

Para reiniciar, se reinicia el timepo, se reinicia los contadoras y se desbloquea el tablero. Se genera una nueva matriz secreta llamando al método inicializar() de la clase tableroSecreto y se vuelven a colocar las imágenes iniciales de los botones.

```
public void reiniciarJuego() {
    // Resetear tiempo
    timer.stop(); // Por si se reinicia sin haber terminado
    tiempo = 0;
    isPrimerClick = true;
    tiempoLabel.setText("Tiempo: " + tiempo);

    // Resetar variables
    minasRestantes = cantMinas;
    minasLabel.setText("Minas: " + minasRestantes);
    faceButton.setIcon(FelizIcon);
    tableroBloqueado = false;

    tableroSecreto.inicializar();
    // Inicializar matriz de botones
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            buttons[i][j].setIcon(TileIcon);
        }
    }
}
```