Práctica 2

Arreglos de Objetos

Programación Orientada a Objetos

Humberto Alejandro Ortega Alcocer (2016630495)

Opción: 7 “Juego de Buscaminas”

12 de noviembre de 2020

Grupo: 2CM1

Introducción

Dentro de los grandes referentes en cuanto a *juegos de computadora* se refiere, uno de los clásicos de todos los tiempos es **Buscaminas**. Y no es para menos, el juego formó parte integral de la familia de sistemas operativos "Windows" diseñados por Microsoft, desde sus orígenes en la década de los 90s hasta hace poco con el lanzamiento de Windows 10 en dónde el juego pasó a ser una parte complementaria y que puede descargarse de forma aislada.
Originalmente, juegos como Buscaminas, Solitario Spider, Carta Blanca (entre otros) se incluían con el sistema operativo a modo de ofrecerle al usuario un entorno para practicar y acostumbrarse al uso de interfaces gráficas mediante el uso del mouse. Este juego, aunque simple en su naturaleza, presenta al usuario frente a un complejo escenario: encontrar todas las *celdas* libres de minas en un tablero dado.
Para llevar a cabo la misión, el juego nos proporciona un número en cada celda no minada indicando el número de minas en las celdas contiguas a esta. Aunque pueda parecer simple, el juego de Buscaminas ha adquirido fama internacional debido a la complejidad *incremental* que se tiene en función del tamaño del tablero. No es lo mismo jugar en un tablero de diez filas y diez columnas con diez minas, que uno de treinta filas y cincuenta columnas con 75 minas. A su vez, el poder aumentar el número de minas arbitrariamente permite tener configuraciones extremadamente complejas como un tablero de diez columnas y diez filas con treinta minas, situación en la cual los números indicativos de las minas en las celdas contiguas dejarán de ser de utilidad y el usuario deberá *adivinar* la posición de las celdas disponibles por lo que fácilmente puede tornarse en una misión imposible.

La ejecución del juego es muy simple, comenzamos con una matriz de botones en la cual deberemos hacer un *salto de fé* seleccionando la primer posición de forma aleatoria. Si nuestro primer intento fue exitoso (es decir que no resultó ser una mina), tendremos una celda con un número entero entre cero y ocho que representará el número de minas en las celdas inmediatamente contiguas a la celda en cuestión. Usando este número como referencia, seleccionaremos la siguiente celda tratando de establecer rutas libres de minas evitando en todo momento *destapar* una celda que pudiera contener una mina.

El juego finaliza una vez que se hayan destapado todas las celdas libres de minas o al destapar, por equivocación, una mina.

El planteamiento de nuestro juego será bien, en palabras del profesor Roberto Tecla:

7.-Juego de busca minas

-Poner imágenes de bombas aleatoriamente en un tablero de botones (similar a como se hace en el caso de los topos).

-En cada botón que no tenga una imagen de bomba contar las bombas que hay en los botones adyacentes y poner dicha cuenta del botón.

Desarrollo

Para el desarrollo de nuestro juego de *Buscaminas*, comenzaremos definiendo el diagrama UML de cada una de las clases que estarán interactuando entre sí. Las clases a considerar para el juego son:

- Buscaminas
- Celda

El diagrama representativo de cada clase se incluye en un comentario al margen superior de cada uno de nuestros archivos de código fuente de forma individual, pero a continuación se presenta un vistazo general:

```
* |-------------------------------------------------------------------|
* |                              Celda                                |
* |-------------------------------------------------------------------|
* | static final int MINA = 10                                        |
* | static final ImageIcon ICONO_MINA = new ImageIcon("mina.png")     |
* | int fila                                                          |
* | int columna                                                       |
* | int valor                                                         |
* |- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
* | int obtenerValor()                                                |
* | void ajustarValor(int)                                            |
* | boolean esMina()                                                  |
* | void descubrir()                                                  |
* | int obtenerFila()                                                 |
* | int obtenerColumna()                                              |
* | static Icon cambiarDimensionesIconoMina(int longitud, int altura) |
* |-------------------------------------------------------------------|
```

Viendo esta información es sencillo de entender el modelado general del juego así como las consideraciones que se han tomado. Para entrar en detalle, comenzaremos analizando la clase **Celda**.

La clase Celda es la encargada de controlar el comportamiento de cada una de las *celdas* dentro de nuestro tablero de juego. Para ello, la clase extiende a la clase *JButton*, por lo que hereda los métodos comunes que empleamos en el manejo de un botón estándar, y añadimos algunos más para poder controlar su comportamiento de forma deliberada. A su vez, incluye dos variables constantes: *ICONO_MINA* y *MINA*, las cuales almacenan el ícono de la mina y el valor con el cual determinamos que una Celda es una mina correspondientemente.

El constructor de la clase recibe como parámetros las coordenadas de la Celda (fila, columna) y una referencia a un *ActionListener* que será aquél que reciba los

eventos emitidos por la Celda en cuestión. La implementación del constructor es la siguiente:

```java
// Constructor de la clase Celda.
public Celda(int filaI, int colI, ActionListener actionListener) {
  this.fila = filaI;
  this.columna = colI;
  this.addActionListener(actionListener);
  this.setText("");
}
```

Como podemos observar el constructor es muy simple y directo.
Por su parte, la clase implementa algunos métodos que nos permiten obtener información y modificar el comportamiento y la apariencia de la Celda en cuestión de forma simple. Un método que es relevante es *cambiarDimensionesIconoMina* el cual es privado, y permite cambiar las dimensiones de la imagen de la mina empleada para las celdas determinadas como minas, de forma dinámica. Su implementación es la siguiente:

```java
// Método para cambiar las dimensiones del ícono de la mina dinámicamente.
private static Icon cambiarDimensionesIconoMina(int longitud, int altura) {
  Image imagen = Celda.ICONO_MINA.getImage(); // Extraemos la imagen.
  Image imagenConNuevasDimensiones = imagen.getScaledInstance(longitud, altura, Image.SCALE_SMOOTH); // La
                                                                                                      // escalamos.
  return new ImageIcon(imagenConNuevasDimensiones); // Regresamos la imagen escalada como un ícono.
}
```

El resto de los métodos en la clase Celda, como ya se mencionó anteriormente, corresponden a métodos *setters* y *getters* con los cuales podemos obtener información de la Celda que se encuentra encapsulada y protegida gracias a los mismos.

Ahora nos enfocaremos en el análisis de la clase **Buscaminas** en la cual se implementa la lógica del juego, así como el control principal de la interfaz gráfica principal del mismo. Uno de los principales objetivos en cualquier juego es poder manipular la complejidad del mismo, por lo que el constructor de la clase recibe tres parámetros para ello: *filas*, *columnas* y *minas*, las cuales corresponden al número de filas, columnas y minas que tendrá nuestro juego.

El constructor de la clase realiza tres tareas fundamentales: inicializar la ventana de la aplicación, inicializar cada una de las celdas, asignar las minas a las celdas y calcular el *valor* para cada una de las celdas en función de las minas contiguas a las mismas. Para calcular la posición de las minas, se utiliza un objeto de la clase *Random* el cual nos permite obtener un número entero dentro de un rango determinado, y se agrega un ciclo *do...while* para verificar que la coordenada generada aleatoriamente no corresponde a una celda previamente marcada como mina. Finalmente, durante el último barrido de la matriz de botones, se realiza el cálculo (mediante un método privado llamado *contarMinasAlrededorDePosicion*) del número de minas alrededor de cada celda dada.

La implementación del constructor se muestra en la siguiente página.

```java
// Constructor de la clase Buscaminas.
public Buscaminas(int filasTablero, int columnasTablero, int minasTablero) {
  this.filas = filasTablero;
  this.columnas = columnasTablero;
  this.numeroMinas = minasTablero;

  // Operaciones para la interfaz gráfica de usuario.
  this.ventana = new JFrame(
      "Buscaminas " + this.filas + "x" + this.columnas + " con " + this.numeroMinas + " minas."); // El título de la
                                                                                                  // ventana es
                                                                                                  // "Buscaminas 10×10
                                                                                                  // con 5 minas.".
  this.ventana.getContentPane().setLayout(new GridLayout(this.filas, this.columnas)); // Usamos un GridLayout para que
                                                                                      // nos permita mostrar la matriz
                                                                                      // de celdas.
  this.ventana.setSize(500, 500); // El tamaño de la ventana lo dejamos fijo en 500×500.

  // Inicializamos nuestra matriz de celdas.
  this.celdas = new Celda[this.filas][this.columnas];

  // Inicializamos cada celda individualmente y la añadimos a nuestra ventana.
  for (int fila = 0; fila < this.filas; fila++) {
    for (int columna = 0; columna < this.columnas; columna++) {
      this.celdas[fila][columna] = new Celda(fila, columna, this);
      this.ventana.getContentPane().add(this.celdas[fila][columna]);
    }
  }

  // Colocamos las minas.
  Random rand = new Random();
  int filaRandom, columnaRandom;
  for (int minasRandom = 0; minasRandom < this.numeroMinas; minasRandom++) {
    // Generamos nuevas coordenadas hasta que sea en un lugar dónde no haya una mina
    // previamente.
    do {
      filaRandom = rand.nextInt(this.filas);
      columnaRandom = rand.nextInt(this.columnas);
    } while (this.celdas[filaRandom][columnaRandom].esMina());

    // Hacemos la celda una mina.
    this.celdas[filaRandom][columnaRandom].ajustarValor(Celda.MINA);
  }

  // Calculamos los números para todas las celdas restantes.
  for (int fila = 0; fila < this.filas; fila++) {
    for (int columna = 0; columna < this.columnas; columna++) {
      // Verificamos si la celda NO es una mina.
      if (!this.celdas[fila][columna].esMina()) {
        // Asignamos la cuenta de minas alrededor de la celda al valor de la celda.
        this.celdas[fila][columna].ajustarValor(this.contarMinasAlrededorDePosicion(fila, columna));
      }
    }
  }

  // Tareas finales para la ventana.
  this.ventana.setVisible(true); // Hacemos visible la ventana.
  this.ventana.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); // Si cerramos la ventana finaliza el
                                                                        // programa.
}
```

A continuación se muestra la implementación del método *contarMinasAlrededorDePosicion* mediante el cual obtenemos el número de minas presentes alrededor de una celda particular:

```java
// Método que cuenta las minas alrededor de una posición determinada en nuestro
// tablero.
private int contarMinasAlrededorDePosicion(int posFila, int posColumna) {
  // Inicializamos nuestro contador en 0.
  int contadorMinas = 0;

  // Iteramos desde la fila superior hasta la fila inferior a la celda actual
  // respetando límites, lo mismo para columnas.
  for (int filaARevisar = (posFila - 1) ⩾ 0 ? (posFila - 1) : 0; filaARevisar < this.filas
      && filaARevisar ⩽ (posFila + 1); filaARevisar++) {
    for (int columnaARevisar = (posColumna - 1) ⩾ 0 ? (posColumna - 1) : 0; columnaARevisar < this.columnas
        && columnaARevisar ⩽ (posColumna + 1); columnaARevisar++) {
      // Verificamos que no sea la celda que estamos revisando.
      if (filaARevisar ≠ posFila || columnaARevisar ≠ posColumna) {
        // Si la celda es una mina, aumentamos el contador.
        if (this.celdas[filaARevisar][columnaARevisar].esMina()) {
          contadorMinas++;
        }
      }
    }
  }

  // Regresamos el valor que contenga nuestro contador.
  return contadorMinas;
}
```

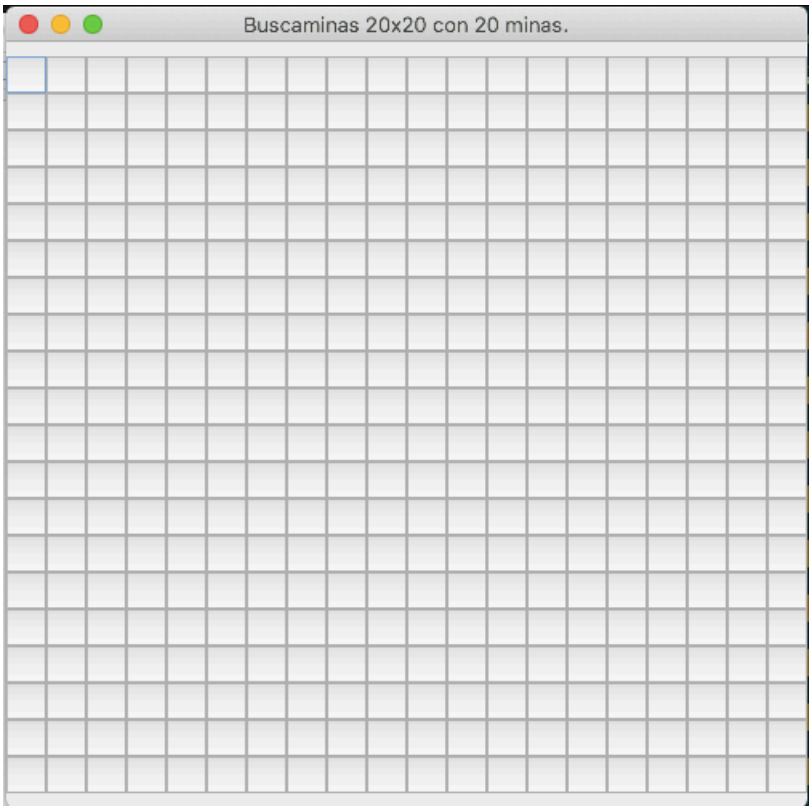
Cabe mencionar que los ciclos corren desde la fila anterior a dónde se encuentra la celda en cuestión (siempre y cuando dicha fila sea mayor a cero) hasta la posición de la fila siguiente a la posición de la celda en cuestión (siempre y cuando dicha fila sea menor al número total de filas) siguiendo la misma lógica con las columnas y llevando un contador simple para determinar el número de minas alrededor de dicha celda.

Finalmente, en el método *actionPerformed* es dónde verificamos qué Celda fue la que realizó la acción, la destapamos independientemente de si es o no una mina y, en caso de serlo, pintamos la celda de color rojo y finalizamos el juego *destapando* todas las celdas del tablero. La implementación es la siguiente:

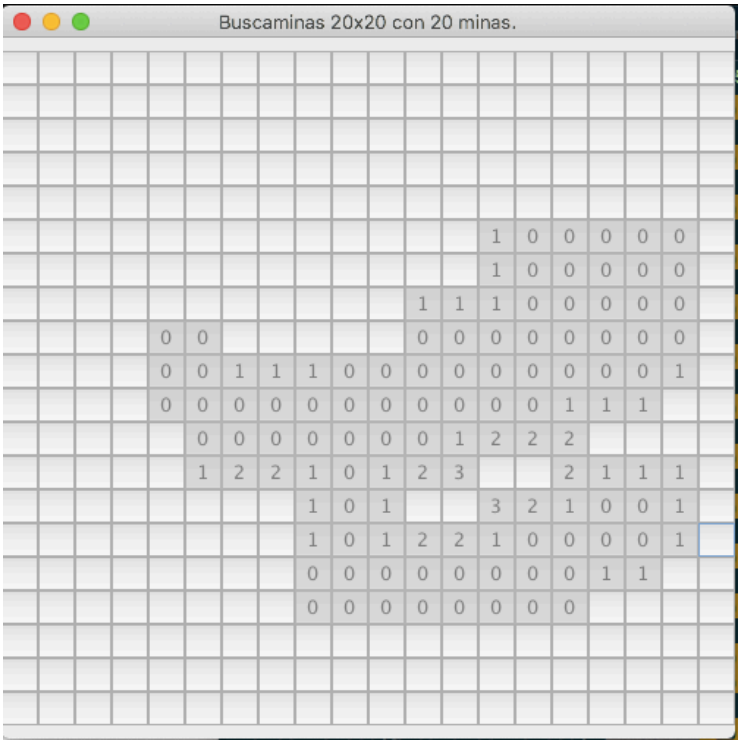
```
// Listener para los eventos de a los que está suscrita la clase.
public void actionPerformed(ActionEvent evento) {
  // Extraemos la celda origen del evento.
  Celda celdaOrigen = (Celda) evento.getSource();

  // Descubrimos la celda (sea cual sea el caso).
  celdaOrigen.descubrir();

  // La volvemos opaca para poder pintarla de un color.
  celdaOrigen.setOpaque(true);

  // Si es una mina, pintamos la celda de rojo y revelamos todo el tablero, en
  // caso contrario, pintamos la
  // celda en verde.
  if (celdaOrigen.esMina()) {
    this.revelarTablero();
    celdaOrigen.setBackground(Color.red);
  } else {
    celdaOrigen.setBackground(Color.lightGray);
  }
}
```

Aunque es simple, con estos elementos nos es posible tener un juego funcional, entretenido y retador de Buscaminas.

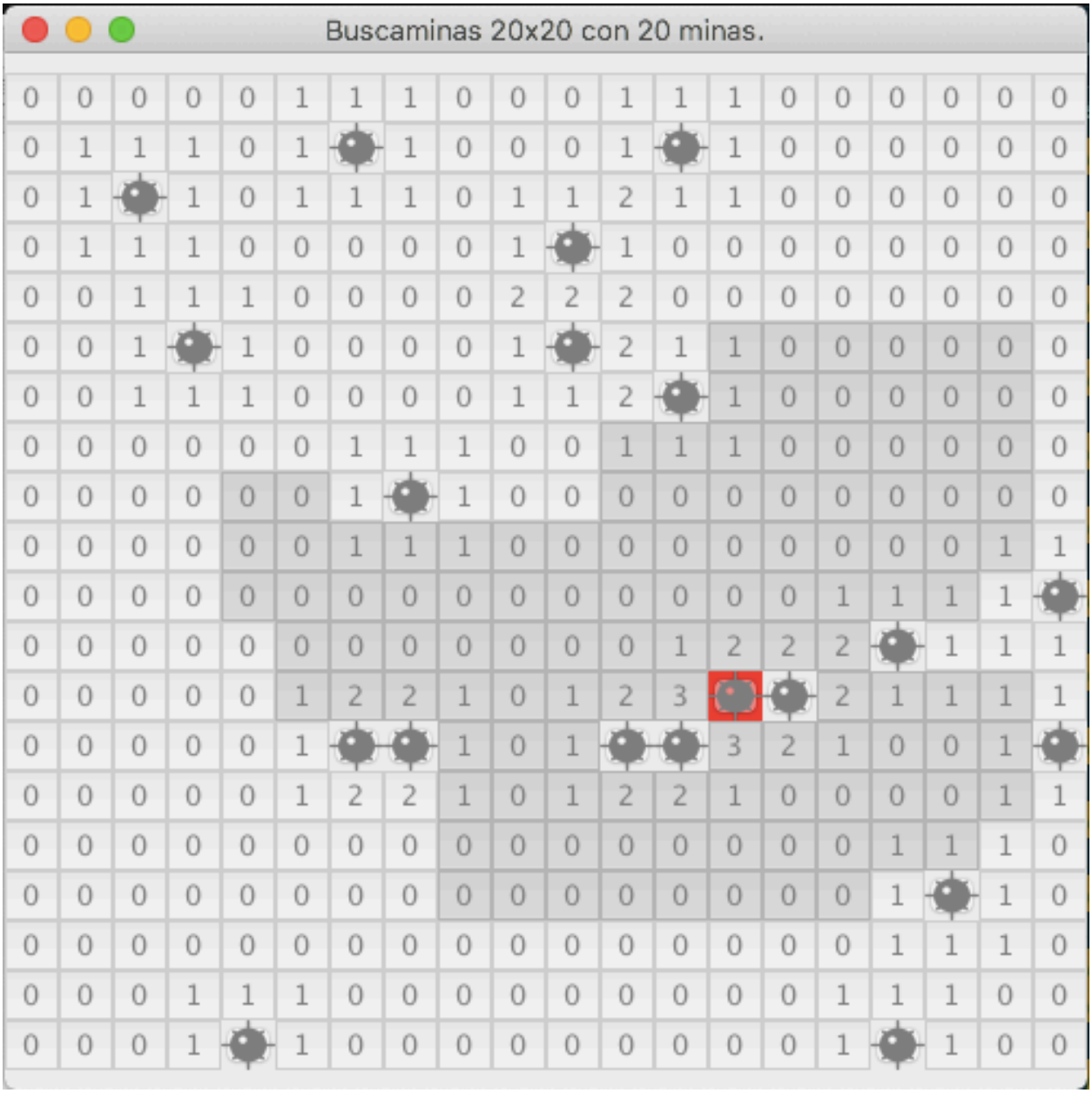
La interfaz gráfica del juego, al iniciar una partida, es la siguiente:



Y después de algunos cuantos tiros, se verá de la siguiente manera:



Y al perder, el juego se mostrará de la siguiente forma:



He decido colocar el fondo de las celdas abiertas por el usuario en un tono gris obscuro, y en color rojo la celda de la mina por la cual perdió el juego.

Conclusión

Si bien el desarrollo de cualquier videojuego conlleva un gran esfuerzo de análisis y programación, en este juego ha resultado muy simple realizar el desarrollo gracias al paradigma de Programación Orientada a Objetos. Pensar en desarrollar el mismo juego en un paradigma de Programación Estructurada o Programación Funcional puede resultar retador al no ser tan evidente y requerir de una gran cantidad de elementos interdependientes para lograr su correcta implementación, sin embargo la simpleza y limpieza en el código que representa el modelado y la abstracción, resultan en una aplicación eficaz y fácil de comprender y extender.

No hay nada más grato para un programador que desarrollar juegos que pueda disfrutar por si mismo. Este juego, en particular, formó parte de mi infancia, en dónde pasé largas horas sin poder entender realmente cómo se jugaba, hasta que un día encontré las instrucciones y desde entonces me volví un jugador compulsivo. Esto es porque, a pesar de que no es uno de los juegos más *modernos* ni *sofisticados*, Buscaminas siempre ha sido un juego rápido, de destreza visual y con un nivel de complejidad difícil de igualar. Si bien jugarlo es muy distinto a desarrollarlo, la realización de esta práctica no ha sido más que una grata experiencia recordando todos esos momentos cuando jugar no requería una conexión a internet ni costosos equipos o configuraciones de alto nivel. Me gustará seguir añadiendo funcionalidades y elementos que, por cuestiones de tiempo para la presente entrega, fueron omitidos, pero lo más importante es que seguiré jugando y divirtiéndome como siempre con este gran juego que, ahora, me tocó a mi entender, analizar e implementar.