Universidad Nacional de General Sarmiento

Programación I

Trabajo Práctico:
Super Elizabeth Sis, Volcano Edition

Profesores: Veronica Moyano y Cesar Niveyro

Grupo: N°1

Integrantes:

Gudiño Alison (*gudinopaniaguacarlyalison@gmail.com*)

Porta Zahira (*zahiraporta2016@gmail.com*)

Vega Victoria (*victoriavvega04@gmail.com*)

Comisión: N°6

Introducción

En el siguiente trabajo se presentará una descripción y explicación del código implementado para poder realizar el juego solicitado en la consigna del TP.

El juego a realizar debía permitir al usuario manejar a la protagonista, la Princesa Elizabeth, y permitirle ir subiendo por los distintos pisos del volcán hasta llegar a la salida mediante las mecánicas de romper bloques y eliminar enemigos.

Descripción

Para poder realizar el juego, creamos las siguientes seis clases para que interactúen entre sí;

- ★ ***Princesa***: la protagonista del juego, el usuario es capaz de manejarla (moverla de izquierda a derecha, hacer que salte, hacer que dispare).
  - ➔ *Variables de instancia*:

```
public class Princesa {
   private double x;
   private double y;
   private Image imagenDerecha;
   private Image imagenIzquierda;
   private int ancho;
   private int alto;
   private int velocidad = 4;
   private double escalaImg = 0.10;
   private double gravedad = 0.5;
   private double velocidadSalto = -12;
   private double velocidadVertical;
   private boolean saltando;
   private boolean mirandoDerecha;
```

  - ➔ *Métodos*:

```
public Princesa(double x, double y) { //constructor de la princesa
       this.x = x;
       this.y = y;
       this.ancho = 45;
```

```
        this.alto = 50;
        this.imagenDerecha =
Herramientas.cargarImagen("assets/princesa.png");
        this.imagenIzquierda =
Herramientas.cargarImagen("assets/princesa_2.png");
        this.velocidadVertical = 0;
        this.saltando = false;
        this.mirandoDerecha = true;
    }
```

```
public void dibujar(Entorno e) { //método para poder dibujar a la princesa
        if(mirandoDerecha) {
              e.dibujarImagen(imagenDerecha, x, y, 0, escalaImg);
        } else {
             e.dibujarImagen(imagenIzquierda, x, y, 0, escalaImg);
        }
     }
```

```
public void moverDerecha(Entorno e) { //método para mover a la derecha
        if (this.x + this.ancho / 2 < e.ancho()) {
            this.x += velocidad;
        }
        this.mirandoDerecha = true;
    }
```

```
public void moverIzquierda() { //método para mover a la izquierda
        if (this.x - this.ancho / 2 >= 0) {
            this.x -= velocidad;
        }
        this.mirandoDerecha = false;
    }
```

```
public void saltar() { //método para permitir saltar a la princesa
        if (!saltando) {  //Solo puede saltar si no está ya en el aire
            velocidadVertical = velocidadSalto;
            saltando = true;
        }
    }
```

```
  public void cancelarSalto() { //método para cancelar el salto
      velocidadVertical =2;
    }
```

```
public void caer(boolean enElAire) { //método para permitir a la princesa
        if (saltando||enElAire) {       caer
            y += velocidadVertical;
            velocidadVertical += gravedad;
        }
    }
```

```
boolean colisionDesdeAbajo(Bloque bloque) {
```

```
//detecta la colisión de la parte superior de la princesa con la parte
inferior del bloque
      return (x + ancho / 2 > bloque.getX() - bloque.getAncho() / 2 &&
                x - ancho / 2 < bloque.getX() + bloque.getAncho() / 2 &&
                y - alto / 2 <= bloque.getY() + bloque.getAlto() / 2 &&
                y - alto / 2 >= bloque.getY());
   }
```

```
boolean colisionDesdeArriba(Bloque bloque) {
// detecta la colisión de la parte inferior de la princesa con la parte
superior del bloque
      return (x + ancho/ 2 > bloque.getX() - bloque.getAncho() / 2 &&
                x - ancho / 2 < bloque.getX() + bloque.getAncho() / 2 &&
                y + ancho / 2 >= bloque.getY() - bloque.getAlto() / 2 &&
                y + ancho / 2 <= bloque.getY()); }
```

```
public boolean colisionaconEnemigo(Enemigo enemigo) {
//detecta si la princesa tuvo esta tocando a el enemigo
       return x < enemigo.getX() + enemigo.getAncho() &&
              x + ancho > enemigo.getX() &&
              y < enemigo.getY() + enemigo.getAlto() &&
              y + alto > enemigo.getY();
       }
```

```
public Bala dispararBala() {
//permite disparar a la princesa y que las balas apunten en la dirección en
la que se disparó
            if( this.mirandoDerecha == true) {
            return new Bala(this.x,this.y,1);
            } else {
                  return new Bala(this.x,this.y,0);
            }
      }
```

★ ***Enemigo***: como su nombre lo indica, es quien intenta dañar a el usuario y hacer que pierda el juego, tiene la capacidad de moverse de izquierda a derecha y dispararle a la princesa, si colisiona con la princesa, se pierde el juego.

➔ *Variables de instancia*:

```
public class Enemigo {
   private double x;
   private double y;
   private Image imagenDerecha;
   private Image imagenIzquierda;
   private int ancho;
   private int alto;
   private double velocidad = 0.5;
   private double gravedad = 0.5;
   private double velocidadX;
   private double velocidadY;
   private double escalaImg = 0.17;
```

```
   private boolean moviendoDerecha;
   boolean colisionDetectada;
```

➔ *Métodos*:

```
public Enemigo(double x, double y) { //constructor de enemigos
       this.x = x;
       this.y = y;
       this.ancho = 45;
       this.alto = 50;
       this.imagenDerecha =
Herramientas.cargarImagen("assets/enemigo_der.png");
       this.imagenIzquierda =
Herramientas.cargarImagen("assets/enemigo_iz.png");
       this.velocidadX = velocidad;
       this.velocidadY = 0;
       this.moviendoDerecha = true;
       this.colisionDetectada = false;
   }
```

```
public void dibujar(Entorno e) { //método para poder dibujar al enemigo
       if (moviendoDerecha) {
           e.dibujarImagen(imagenDerecha, x, y-12, 0, escalaImg);
       } else {
           e.dibujarImagen(imagenIzquierda, x, y-12, 0, escalaImg);
       }
   }
```

```
public void mover(Entorno e, Bloque[] bloques) {
//permite a el enemigo moverse de izquierda a derecha y cambiar de direccion
si llega a los bordes de la pantalla

       velocidadY += gravedad;
       y += velocidadY;
       for (Bloque bloque : bloques) {
           if (bloque != null && colisionDesdeArriba(bloque)) {
               velocidadY = 0;
               y = bloque.getY() - bloque.getAlto() / 2 - alto / 2;
               colisionDetectada = true;
           }
       }
       if (x + ancho / 2 >= e.ancho()) {
           x = e.ancho() - ancho / 2;
           cambiarDireccion();
       } else if (x - ancho / 2 <= 0) {
           x = ancho / 2;
           cambiarDireccion();
       }
       if (moviendoDerecha) {
           x += velocidadX;
       } else {
           x -= velocidadX;
```

```
        }

    }
```

```
public BombaEnemigo dispararBomba() {
//permite a el enemigo disparar
            if( this.moviendoDerecha == true) {
            return new BombaEnemigo(this.x,this.y,1);
            } else {
                  return new BombaEnemigo(this.x,this.y,0);
            }
      }
```

```
public void cambiarDireccion() { //permite cambiar de direcciona al enemigo
       moviendoDerecha = !moviendoDerecha; }
```

```
private boolean colisionDesdeArriba(Bloque bloque) {
//verifica si el enemigo esta arriba de un bloque
       return (x + ancho / 2 > bloque.getX() - bloque.getAncho() / 2 &&
               x - ancho / 2 < bloque.getX() + bloque.getAncho() / 2 &&
               y + alto / 2 >= bloque.getY() - bloque.getAlto() / 2 &&
               y + alto / 2 <= bloque.getY());
   }

//verifica si el enemigo fue atacado por una bala de la princesa
public boolean colisionaCon(Bala bala) {
       return (bala.getX() + bala.getAncho() / 2 > x - ancho / 2 &&
               bala.getX() - bala.getAncho() / 2 < x + ancho / 2 &&
               bala.getY() + bala.getAlto() / 2 > y - alto / 2 &&
               bala.getY() - bala.getAlto() / 2 < y + alto / 2);
   }
```

★ ***Bloque***: los bloques en el juego sirven como los distintos pisos, para ganar, la princesa tiene que subir hasta la cima, tienen la capacidad de romperse si la princesa salta por debajo de ellos, pero hay bloques especiales que no pueden ser destruidos por nada.

➔ *Variables de instancia*:

```
public class Bloque {
   private double x;
   private double y;
   private double ancho;
   private double alto;
   boolean indestructible;
   private Image destruible;
   private Image indestruible;
```

➔ *Métodos*:

```
  public Bloque(double x, double y, boolean indestructible) {
       this.x = x;
       this.y = y;
       this.ancho = 52;
```

```
        this.alto = 52;
        this.indestructible = indestructible;
        this.destruible =
Herramientas.cargarImagen("assets/destruible.png");
        this.indestruible =
Herramientas.cargarImagen("assets/indestruible.png");
    }
```

```
 public void dibujar(Entorno e) { //permite dibujar el bloque
      if(indestructible) {
           e.dibujarImagen(indestruible, x, y, 0,0.1);
      } else {
             e.dibujarImagen(destruible, x, y, 0,0.1);
      }
    }
```

★ ***Bala***: es lo que la princesa dispara al realizar su ataque especial, al colisionar con un enemigo lo elimina, también elimina los ataques de los enemigos y si sale fuera de la pantalla es eliminada automáticamente. Solo puede haber una bala en la pantalla.

➔ *Variables de instancia*:

```
public class Bala {
   private double x;
   private double y;
   private Image imagenDerecha;
   private Image imagenIzquierda;
   private int VELOCIDAD = 5;
   private final double ESCALA_IMAGEN = 0.10;
   private final int ancho = 10;
   private final int alto = 10;
   int direccion;
```

➔ *Métodos*:

```
public Bala(double x, double y, int d) {
        this.x = x;
        this.y = y;
        this.imagenDerecha =
Herramientas.cargarImagen("assets/bala_der.png");
        this.imagenIzquierda =
Herramientas.cargarImagen("assets/bala_iz.png");
        direccion=d;
    }
```

```
public void dibujar(Entorno entorno) { // Dibuja la bala en el entorno
        if(direccion==1) {
            entorno.dibujarImagen(imagenDerecha, x, y, 0, ESCALA_IMAGEN);
        } else {
            entorno.dibujarImagen(imagenIzquierda, x, y, 0, ESCALA_IMAGEN);
        }
```

```
    }
```

```
// Mueve la bala en la dirección actual
    public void mover() {
        if (direccion==1) {
                this.x += VELOCIDAD;
            } else {
                this.x -= VELOCIDAD;
            }
    }
```

```
//Verifica si la bala está fuera de la pantalla
    public boolean fueraDePantalla(Entorno e) {
        return (x >e.ancho()|| x < 0);
    }
```

```
    // Verifica si la bala colisiona con un enemigo
    public boolean colisionaCon(Enemigo enemigo) {
        return (x + ancho / 2 > enemigo.getX() - enemigo.getAncho() / 2 &&
                x - ancho / 2 < enemigo.getX() + enemigo.getAncho() / 2 &&
                y + alto / 2 > enemigo.getY() - enemigo.getAlto() / 2 &&
                y - alto / 2 < enemigo.getY() + enemigo.getAlto() / 2);
    }
```

```
public boolean colisionaCon(BombaEnemigo bomba) {
        return (x + ancho / 2 > bomba.getX() - bomba.getAncho() / 2 &&
                x - ancho / 2 < bomba.getX() + bomba.getAncho() / 2 &&
                y + alto / 2 > bomba.getY() - bomba.getAlto() / 2 &&
                y - alto / 2 < bomba.getY() + bomba.getAlto() / 2);
    }
```

★ ***BombaEnemigo***: es lo que el enemigo dispara para intentar eliminar a la princesa, si una bomba del enemigo alcanza a la princesa, el usuario perderá el juego. Si una bomba es alcanzada por una de las balas de la princesa, la misma es eliminada, y si sale de la pantalla también es eliminada automáticamente.

➔ *Variables de instancia*:

```
public class BombaEnemigo {
        private double x;
           private double y;
           private Image imagenDerecha;
           private Image imagenIzquierda;
           private int velocidad = 1;
           private final double escalaImagen = 0.20;
           private final int ancho = 5;
           private final int alto = 5;
           int direccion;
```

➔ *Métodos*:

```
 public BombaEnemigo (double x, double y, int d) {
```

```
                this.x = x;
                this.y = y;
                this.imagenDerecha =
Herramientas.cargarImagen("assets/flecha_der.png");
                this.imagenIzquierda =
Herramientas.cargarImagen("assets/flecha_iz.png");
                direccion=d;
            }
```

```
// Dibuja la bomba en el entorno
            public void dibujar(Entorno entorno) {
                if(direccion==1) {
                    entorno.dibujarImagen(imagenDerecha, x, y, 0,
escalaImagen);
                } else {
                    entorno.dibujarImagen(imagenIzquierda, x, y, 0,
escalaImagen);
                }
            }
```

```
// Mueve la bala en la dirección actual
            public void mover() {
                if (direccion==1) {
                        this.x += velocidad;
                    } else {
                        this.x -= velocidad;
                    }
            }
```

```
//Verifica si la bala está fuera de la pantalla
            public boolean fueraDePantalla(Entorno e) {
                return (x >e.ancho()|| x < 0 );
            }
```

```
// Verifica si la bomba coliciona con la princesa
            public boolean colisionaCon(Princesa princesa) {
                return (x + ancho / 2 > princesa.getX() - princesa.getAncho()
/ 2 &&
                        x - ancho / 2 < princesa.getX() + princesa.getAncho()
/ 2 &&
                        y + alto / 2 > princesa.getY() - princesa.getAlto() /
2 &&
                        y - alto / 2 < princesa.getY() + princesa.getAlto() /
2);
            }
```

```
//verifica si la bomba coliciona con la bala de la princesa
            public boolean colisionaCon(Bala bala) {
                return (x + ancho / 2 > bala.getX() - bala.getAncho() / 2 &&
                        x - ancho / 2 < bala.getX() + bala.getAncho() / 2 &&
                        y + alto / 2 > bala.getY() - bala.getAlto() / 2 &&
                        y - alto / 2 < bala.getY() + bala.getAlto() / 2);
```

```
        }
```

★ ***Gatito***: el gatito, la mascota de la princesa, sirve como un indicador de a donde tiene que dirigirse el usuario si desea ganar el juego. Si la princesa lo toca, el usuario gana el juego.

➔ *Variables de instancia*:

```
public class Gatito {
      private double x;
   private double y;
   private Image gato;
```

➔ *Métodos*:

```
public Gatito(double x, double y) { //constructor gatito
      this.x=x;
      this.y=y;
      this.gato=Herramientas.cargarImagen("assets/gatito.png");
   }
```

```
 public void dibujar(Entorno e) {  //método para poder dibujar al gatito
            e.dibujarImagen(gato, x, y, 0, 0.085);
       }
```

```
 public boolean salvado(Princesa p) { //verifica si la princesa está en la
misma posición que el gatito y si es así, retorna true, si no, false.
      if(p.getX()==x&&p.getY()+8==y) {
            return true;
      }
      return false;

   }
```

★ ***Implementación de las distintas clases en la clase Juego***: La clase Juego es la clase principal del programa que maneja la lógica del juego, incluyendo la inicialización de los objetos del juego, la actualización del estado del juego en cada "tick" del reloj del entorno, la detección de colisiones, y la gestión de las condiciones de victoria y derrota. Esta clase extiende InterfaceJuego, que proporciona el marco necesario para la ejecución del juego en el entorno proporcionado.

➔ *Variables de instancia*:

```
 public class Juego extends InterfaceJuego {
    private Entorno entorno;// El objeto Entorno que controla el tiempo
y otros
    private ImageIcon fondo;

    private ImageIcon fondo_Inicio;
```

```
    private ImageIcon fondo_Final;

    private boolean enJuego;

    private Princesa princesa;

    private Bloque[] bloques;

    private Enemigo[] enemigos;

    private Bala balaP; //balas princesa

    private Gatito gatito;

    private int puntos,eliminados;

    private BombaEnemigo[] bombasEnemigos; // Bombas lanzadas por los
enemigos

    private boolean juegoTerminado; // Variable para controlar el
estado del juego

    private boolean juegoGanado; // Variable para controlar el estado
del juego
```

➔ *Métodos*:

```
private void iniciarEnemigos() { //inicia dos enemigos por cada fila de
bloques
            int enemigosPorPiso = 2;
            for (int i=0; i<enemigos.length; i++) {
                  int piso = i / enemigosPorPiso;
                  double x =100+(i%enemigosPorPiso)*400;
                  double y = 100+(piso*150);
                  enemigos[i]=new Enemigo(x,y);
          }
   }
```

```
private void iniciarBloques(int filas) {
//inicia dos tipos de bloques en cada fila, los destructibles y los
indestructibles.
            int xInicial = 24;
          int yInicial = entorno.alto() - xInicial;
          int yAltura = 150;
          int totalFilas = filas;
          int totalBloquesPorFila = bloques.length/totalFilas;
```

```
          int desplazamiento = 3; // desplazamiento para cada fila
          for (int fila = 0; fila < totalFilas; fila++) {
              int yFila = yInicial - (fila * yAltura);
              int inicioIndestructibles = (fila * desplazamiento) %
(totalBloquesPorFila - 4);
              for (int i = 0; i < totalBloquesPorFila; i++) {
                  int x = xInicial + (i * 50); //separacion entre bloques
                  if(fila==0) {
                  this.bloques[fila * totalBloquesPorFila+i] = new
Bloque(x,yFila,false);
                  }
                  else if (i >= inicioIndestructibles && i <
inicioIndestructibles + 4) {
                      this.bloques[fila * totalBloquesPorFila + i] = new
Bloque(x, yFila, true); // Bloque indestructible
                  } else {
                      this.bloques[fila * totalBloquesPorFila + i] = new
Bloque(x, yFila, false); // Bloque destructible
                  }
                  }
            } }
```

```
private boolean princesaEnElAire() { //verifica si la princesa esta en el
aire, de ser así retorna true
       for (int i = 0; i < bloques.length; i++) {
           if (bloques[i] != null &&
               princesa.getX() + princesa.getAncho() / 2 > bloques[i].getX()
- bloques[i].getAncho() / 2 &&
               princesa.getX() - princesa.getAncho() / 2 < bloques[i].getX()
+ bloques[i].getAncho() / 2 &&
               princesa.getY() + princesa.getAlto() / 2 <= bloques[i].getY()
- bloques[i].getAlto() / 2 &&
               princesa.getY() + princesa.getAlto() / 2 >= bloques[i].getY()
- bloques[i].getAlto() / 2 - 5) {
               return false;
           }
       }
       return true;
   }
```

```
  private void dibujarPuntuacion() { //método utilizado para dibujar en
pantalla la puntuación actual del jugador
            entorno.cambiarFont("Minecraft",20,Color.white);
            entorno.escribirTexto("Puntos: " + this.puntos, 20, 585);
            entorno.escribirTexto("Enemigos eliminados: " + this.eliminados,
150, 585);

      }
```

```
      private void dibujarFondo() { //método para dibujar la imagen del
fondo del juego
entorno.dibujarImagen(fondo.getImage(),entorno.ancho()/2,entorno.alto()/2,0);
```

```
    }
```

En el método tick, se incluyó;

```
        // Hace que la princesa caiga si está en el aire
        princesa.caer(princesaEnElAire());

        // Dibuja la princesa en la pantalla
        princesa.dibujar(entorno);

        // Dibuja el gatito en la pantalla
        gatito.dibujar(entorno);

// Movimiento de la princesa a la derecha
        if (entorno.estaPresionada(entorno.TECLA_DERECHA)) {
            princesa.moverDerecha(entorno);
        }

              // Movimiento de la princesa a la izquierda
        if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) {
            princesa.moverIzquierda();
        }

        // Salto de la princesa
        if (entorno.sePresiono('x')) {
            princesa.saltar();
        }

        // Disparo de bala por la princesa
        if (entorno.sePresiono('c') && balaP == null) {
            balaP = princesa.dispararBala();
        }

// Colisión con bloques
        for (int i = 0; i < bloques.length; i++) {
            if (bloques[i] != null && !bloques[i].indestructible &&
princesa.colisionDesdeAbajo(bloques[i])) {
                bloques[i] = null;  // Elimina el bloque si no es
indestructible y hay colisión desde abajo
                princesa.cancelarSalto();  // Cancela el salto de la
princesa
            }

  // Dibuja los bloques
            if (bloques[i] != null) {
                this.bloques[i].dibujar(entorno);

                // Cancela el salto si colisiona con un bloque
indestructible desde abajo
                if (bloques[i].indestructible &&
princesa.colisionDesdeAbajo(bloques[i])) {
```

```
                    princesa.cancelarSalto();
                }
```

```
            // Detiene la caída si colisiona con un bloque desde arriba
                if (princesa.colisionDesdeArriba(bloques[i])) {
                    princesa.detenerCaida(bloques[i].getY() - 29);
                }
            }
        }
```

```
// Movimiento y dibujado de los enemigos
        for (int i = 0; i < enemigos.length; i++) {
            if (enemigos[i] != null) {
                enemigos[i].mover(entorno, bloques);
                enemigos[i].dibujar(entorno);

                // Cada enemigo dispara una bomba si no tiene una activa
                if (bombasEnemigos[i] == null) {
                    bombasEnemigos[i] = enemigos[i].dispararBomba();
                }
```

```
 // Verifica colisión con la princesa
                for (Enemigo enemigo : enemigos) {
                    if (enemigo != null &&
princesa.colisionaconEnemigo(enemigo)) {
                        juegoTerminado = true;
                        return;
                    }
                }
            } }
        // Movimiento y dibujado de la bala de la princesa
        if (balaP != null) {
            balaP.mover();
            balaP.dibujar(entorno);
```

```
       // Elimina la bala si sale de la pantalla
            if (balaP.fueraDePantalla(entorno)) {
                balaP = null;
            }
        }
```

```
// Verifica colisión de la bala con los enemigos
        if (balaP != null) {
            for (int j = 0; j < enemigos.length; j++) {
                Enemigo enemigo = enemigos[j];
                if (balaP != null && enemigo != null &&
balaP.colisionaCon(enemigo)) {
                    enemigos[j] = null;  // Elimina al enemigo
                    balaP = null;  // Elimina la bala
                this.puntos += 2; //suma dos puntos por enemigo eliminado
                this.eliminados += 1; //cuenta los enemigos eliminados
```

```
                    }
                }
            }
```

```
// Dibuja la puntuación en la pantalla
            dibujarPuntuacion();
```

```
// Verifica si la princesa ha salvado al gatito
            if (gatito.salvado(princesa)) {
                juegoGanado = true;
            }
            // Movimiento y dibujado de las bombas de los enemigos
            for (int i = 0; i < bombasEnemigos.length; i++) {
                if (bombasEnemigos[i] != null) {
                    bombasEnemigos[i].mover();
                    bombasEnemigos[i].dibujar(entorno);
                    // Elimina la bomba si sale de la pantalla
                    if (bombasEnemigos[i].fueraDePantalla(entorno)) {
                        bombasEnemigos[i] = null;
                    }
                }
            }
```

```
// Verifica colisión de la bala de la princesa con las bombas enemigas
            for (int a = 0; a < bombasEnemigos.length; a++) {
                if (bombasEnemigos[a] != null && balaP != null &&
balaP.colisionaCon(bombasEnemigos[a])) {
                    bombasEnemigos[a] = null;  // Elimina la bomba
                    balaP = null;  // Elimina la bala
                } }
```

```
 // Verifica colisión de la princesa con los enemigos
            for (Enemigo enemigo : enemigos) {
                if (enemigo != null && princesa.colisionaconEnemigo(enemigo))
{
                    juegoTerminado = true;
                    return;
                }
            }
```

```
// Verifica colisión de la princesa con las bombas enemigas
            for (BombaEnemigo bomba : bombasEnemigos) {
                if (bomba != null && bomba.colisionaCon(princesa)) {
                    juegoTerminado = true;
                    return;
                           }
                    }
                    }
```

★ ***Extras***: Se añadió un menú de inicio, una pantalla de derrota y una pantalla de victoria, además de la opción de reiniciar el juego con la letra “v” y salir del juego con la letra “s”.

```
	private void dibujarFondoInicio() {

entorno.dibujarImagen(fondo_Inicio.getImage(),entorno.ancho()/2,entorno.alto(
)/2,0);
	}

	private void dibujarFondoFinal() {

entorno.dibujarImagen(fondo_Final.getImage(),entorno.ancho()/2,entorno.alto()
/2,0);
	}

	private void dibujarMenuInicio() {
	 dibujarFondoInicio();
	 entorno.cambiarFont("Venice Classic", 60, Color.YELLOW);
	 entorno.escribirTexto("Super Elizabeth Sis", entorno.ancho() / 2 -
220, entorno.alto() / 2 - 70);
	 entorno.escribirTexto("Volcano Edition", entorno.ancho() / 2 - 190,
entorno.alto() / 2 - 10);
	 entorno.cambiarFont("Minecraft", 18, Color.YELLOW);
	 entorno.escribirTexto("Empezar (Espacio)", entorno.ancho() / 2 - 90,
entorno.alto() / 2 + 50);
	 entorno.escribirTexto("Salir (S)", entorno.ancho() / 2 - 43,
entorno.alto() / 2 + 100);
   }

   private void dibujarPuntuacion() {
	    entorno.cambiarFont("Minecraft",20,Color.white);
	    entorno.escribirTexto("Puntos: " + this.puntos, 20, 585);
	    entorno.escribirTexto("Enemigos eliminados: " + this.eliminados,
150, 585);

	}

	private void dibujarVictoria() {
	      dibujarFondoFinal();
	      entorno.cambiarFont("Venice Classic", 60, Color.YELLOW);
	    entorno.escribirTexto("Ganaste", entorno.ancho()/2 - 95,
entorno.alto()/2);
	    entorno.cambiarFont("Minecraft", 18, Color.YELLOW);
	    entorno.escribirTexto("reiniciar (V)", entorno.ancho() / 2 - 43,
entorno.alto() / 2 + 100);
	    entorno.escribirTexto("Salir (S)", entorno.ancho() / 2 - 43,
entorno.alto() / 2 + 120);
	    if (entorno.sePresiono('v')) {
	     reiniciarJuego();
	   }
	   if(entorno.sePresiono('s')) {
	        salir();
	   }
```

```
      }

      private void dibujarDerrota() {
            entorno.dibujarRectangulo(entorno.ancho() / 2, entorno.alto() /
2, entorno.ancho(), entorno.alto(), 0, Color.BLACK);
            entorno.cambiarFont("Venice Classic", 60, Color.RED);
          entorno.escribirTexto("Perdiste", entorno.ancho()/2 - 95,
entorno.alto()/2);
          entorno.cambiarFont("Minecraft", 18, Color.RED);
          entorno.escribirTexto("reiniciar (V)", entorno.ancho() / 2 - 43,
entorno.alto() / 2 + 100);
          entorno.escribirTexto("Salir (S)", entorno.ancho() / 2 - 43,
entorno.alto() / 2 + 120);
          if (entorno.sePresiono('v')) {
           reiniciarJuego();
       }
       if(entorno.sePresiono('s')) {
            salir();
       }
      }

      public void iniciarJuego() {
       this.enJuego = true;
   }
   public void salir() {
       System.exit(0);
   }

      private void reiniciarJuego() {
       this.princesa = new Princesa(entorno.ancho() / 2, entorno.alto() -
80);
       this.gatito = new Gatito(80, 80);
       this.bloques = new Bloque[64];
       this.iniciarBloques(4);
       this.enemigos = new Enemigo[8];
       this.iniciarEnemigos();
       this.bombasEnemigos = new BombaEnemigo[enemigos.length];
       this.balaP = null;
       this.puntos = 0;
       this.eliminados = 0;
       this.juegoTerminado = false;
       this.juegoGanado = false;
      }
```

En el método tick se añadió una condición que diga si el juego ya empezó (empieza al presionar la tecla espacio) si todavía no empezó entonces se muestra la pantalla de inicio, si ya empezó se inicia el juego.

Problemas y soluciones

Algunos de los problemas que fuimos encontrando a lo largo del desarrollo fueron, por ejemplo;

- ★ Problemas con la organización del código. Al principio, todas los métodos de detección de colisiones estaban en la clase juego, en un comienzo, cuando en la clase juego no había tantas cosas, esto no presentaba un problema, pero a medida que se fue agregando más código, decidimos mejor ponerlas dentro de cada clase, para que no sea confuso.
- ★ Caída de princesa. Uno de los mayores problemas que tuvimos fue que cuando la princesa dejaba null un bloque si la misma caminaba por encima de este, no caía.

Para solucionar esto implementamos dos métodos; *caer(enElAire*) y *princesaEnElAire()*, princesaEnElAire recorre el arreglo de bloques creados en el juego, si los mismos no son null y además ocurre una colisión con ellos, retorna false (la princesa **no** está en el aire) si hay una colisión con los bloques, pero los mismos son null o si simplemente no hay una colisión en el momento, retorta true(la princesa **si** está en el aire). Este booleano es utilizado en el método caer en el cual, si la princesa está saltando o si está en el aire, le permite caer

- ★ Iteración y colisiones complejas. Otro problema que tuvimos fueron las colisiones entre princesa, enemigo, bloque, balas y bombas porque se hizo un uso extensivo de las coordenadas y dimensiones de los objetos para determinar si ocurrían colisiones y cómo debían manejarse. Incorporamos las verificaciones de colisiones directamente dentro del bucle principal del juego (método tick()). Esto simplificó la estructura del código.

Conclusiones

El desarrollo de este juego ha sido un desafío significativo que ha puesto a prueba nuestras habilidades de programación y nuestra capacidad para resolver problemas. Desde la implementación de la lógica del juego hasta la gestión de las colisiones y la creación de los diferentes estados del juego, cada etapa requirió un gran esfuerzo y dedicación.
En resumen, aunque no fue fácil desarrollar este juego, el esfuerzo invertido ha valido la pena, y estamos orgullosas del producto final que hemos creado. Este proyecto ha sido una experiencia enriquecedora que ha contribuido significativamente a nuestro crecimiento como programadoras.