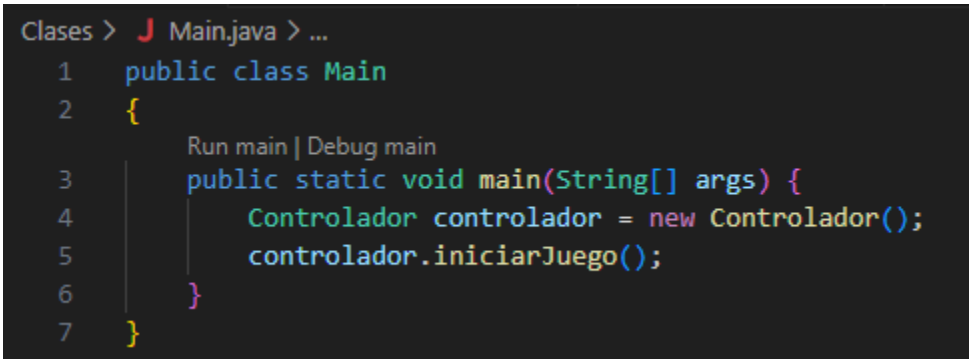
Apéndice 2: Capturas de pantalla del código

Main.java:



Mapa.java:

```java
public class Mapa
{
    private final String[][] arrayMapa1 = {
        {"X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X"},
        {"X"," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," ","X"},
        {"X"," ","X","X","X","X"," ","X"," ","X","X","X"," ","X","X"," ","X","X"," ","X","X","X"," ","X"," ","X","X","X","X"," ","X"},
        {"X"," ","X","S"," "," "," ","X"," "," "," "," "," ","X","X"," ","X","X"," "," "," "," "," ","X"," "," "," ","S","X"," ","X"},
        {"X"," ","X"," ","X","X"," ","X","X","X"," ","X"," "," "," "," "," "," "," ","X"," ","X","X","X"," ","X","X"," ","X"," ","X"},
        {"X"," ","X"," ","X","X"," ","X"," "," "," ","X"," ","X","X","X","X","X"," ","X"," "," "," ","X"," ","X","X"," ","X"," ","X"},
        {"X"," "," "," "," "," "," ","X"," ","X","X","X"," ","X"," "," "," ","X"," ","X","X","X"," ","X"," "," "," "," "," "," ","X"},
        {"X"," ","X","X","X","X"," "," "," "," "," "," "," ","X","1","X","2","X"," "," "," "," "," "," "," ","X","X","X","X"," ","X"},
        {"X"," "," "," "," "," "," ","X","X","X","X","X"," "," "," ","X"," "," "," ","X","X","X","X","X"," "," "," "," "," "," ","X"},
        {"X"," ","X","X","X","X"," "," "," "," "," "," "," ","X","3","X","4","X"," "," "," "," "," "," "," ","X","X","X","X"," ","X"},
        {"X"," "," "," "," "," "," ","X"," ","X","X","X"," ","X"," "," "," ","X"," ","X","X","X"," ","X"," "," "," "," "," "," ","X"},
        {"X"," ","X"," ","X","X"," ","X"," "," "," ","X"," ","X","X","X","X","X"," ","X"," "," "," ","X"," ","X","X"," ","X"," ","X"},
        {"X"," ","X"," ","X","X"," ","X","X","X"," ","X"," "," "," ","P"," "," "," ","X"," ","X","X","X"," ","X","X"," ","X"," ","X"},
        {"X"," ","X","S"," "," "," ","X"," "," "," "," "," ","X","X"," ","X","X"," "," "," "," "," ","X"," "," "," ","S","X"," ","X"},
        {"X"," ","X","X","X","X"," ","X"," ","X","X","X"," ","X","X"," ","X","X"," ","X","X","X"," ","X"," ","X","X","X","X"," ","X"},
        {"X"," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," ","X"},
        {"X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X","X"}
    };

 private final String[][] arrayMapaGameOver = {
     {" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X","X","X","X"," "," ","X","X"," "," ","X"," "," "," ","X"," ","X","X","X","X"," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," "," "," ","X"," "," ","X"," ","X","X"," ","X","X"," ","X"," "," "," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," "," "," ","X"," "," ","X"," ","X"," ","X"," ","X"," ","X"," "," "," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," ","X","X"," ","X","X","X","X"," ","X"," "," "," ","X"," ","X","X","X","X"," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X"," "," "," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X"," "," "," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X","X","X","X"," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X","X","X","X"," "," "," "," "," "," "},
     {" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "},
     {" "," "," "," "," "," ","X","X"," "," ","X"," "," "," ","X"," ","X","X","X","X"," ","X","X","X"," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X"," "," "," "," ","X"," "," ","X"," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X"," "," "," "," ","X"," "," ","X"," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X","X","X","X"," ","X","X","X"," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," ","X"," "," "," ","X"," ","X"," "," "," "," ","X"," ","X"," "," "," "," "," "," "," "},
     {" "," "," "," "," ","X"," "," ","X"," "," ","X"," ","X"," "," ","X"," "," "," "," ","X"," "," ","X"," "," "," "," "," "," "},
     {" "," "," "," "," "," ","X","X"," "," "," "," ","X"," "," "," ","X","X","X","X"," ","X"," "," ","X"," "," "," "," "," "," "},
     {" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "}
 };

 public String[][] getArrayMapa1() {
     return arrayMapa1;
 }

 public String[][] getArrayMapaGameOver() {
     return arrayMapaGameOver;
 }
```

Pacman.java:

```java
public class Pacman
{
    public static String direccionDeseada = "arriba";
    public static String direccionEnEspera = "arriba";

    private String[] direcciones = {"arriba", "abajo", "derecha", "izquierda"};
    public static String direccion;

    public int posicionX;
    public int posicionY;

    public int[] coordenadasPacman = {posicionY, posicionX};

    private Controlador controlador;

    String[][] mapa;

    // Constructor que inicializa el controlador y el mapa
    public Pacman(Controlador controlador) {
        this.controlador = controlador;
        this.mapa = controlador.getMapa().getArrayMapa1();
    }
```

```java
// Método para mover a Pacman en la dirección deseada
public void mover(String direccionDeseada) {
    if(Controlador.juegoIniciado) {
        for(int i = 0; i < mapa.length; i++) {
            for(int j = 0; j < mapa[i].length; j++) {
                if(mapa[i][j].contains("P")) {
                    switch(direccionDeseada) {
                        case "arriba":
                            if(mapa[i - 1][j].contains(" ") || mapa[i - 1][j].contains("S") || mapa[i - 1][j].contains("C") || mapa[i - 1][j].contains("1")
                                || mapa[i - 1][j].contains("2") || mapa[i - 1][j].contains("3") || mapa[i - 1][j].contains("4")) {
                                controlador.verificarMuerte();
                                if(mapa[i - 1][j].contains(" ")) {
                                    controlador.incrementarPuntos();
                                } else if(mapa[i - 1][j].contains("S")) {
                                    controlador.SuperIncrementarPuntos();
                                }
                                mapa[i - 1][j] += "P";
                                mapa[i][j] = mapa[i][j].replaceAll(" ", "C");
                                mapa[i][j] = mapa[i][j].replaceAll("S", "C");
                                mapa[i][j] = mapa[i][j].replaceAll("P", "");
                                direccion = "arriba";
                                posicionY = i - 1;
                                posicionX = j;
                                coordenadasPacman[0] = posicionY;
                                coordenadasPacman[1] = posicionX;
                                return;
                            } else if(mapa[i - 1][j].contains("X")) {
                                direccionDeseada = direccion;
                                direccionEnEspera = "arriba";
                            }
                            break;
```

```
                case "abajo":
                    if(mapa[i + 1][j].contains(" ") || mapa[i + 1][j].contains("S") || mapa[i + 1][j].contains("C") || mapa[i + 1][j].contains("1")
                        || mapa[i + 1][j].contains("2") || mapa[i + 1][j].contains("3") || mapa[i + 1][j].contains("4")) {
                        controlador.verificarMuerte();
                        if(mapa[i + 1][j].equals(" ")) {
                            controlador.incrementarPuntos();
                        } else if(mapa[i + 1][j].equals("S")) {
                            controlador.SuperIncrementarPuntos();
                        }
                        mapa[i + 1][j] += "P";
                        mapa[i][j] = mapa[i][j].replaceAll(" ", "C");
                        mapa[i][j] = mapa[i][j].replaceAll("S", "C");
                        mapa[i][j] = mapa[i][j].replaceAll("P", "");
                        direccion = "abajo";
                        posicionY = i + 1;
                        posicionX = j;
                        coordenadasPacman[0] = posicionY;
                        coordenadasPacman[1] = posicionX;
                        return;
                    } else if(mapa[i + 1][j].contains("X")) {
                        direccionDeseada = direccion;
                        direccionEnEspera = "abajo";
                    }
                    break;
                case "derecha":
                    if(mapa[i][j + 1].contains(" ") || mapa[i][j + 1].contains("S") || mapa[i][j + 1].contains("C") || mapa[i][j + 1].contains("1")
                        || mapa[i][j + 1].contains("2") || mapa[i][j + 1].contains("3") || mapa[i][j + 1].contains("4")) {
                        controlador.verificarMuerte();
                        if(mapa[i][j + 1].equals(" ")) {
                            controlador.incrementarPuntos();
                        } else if(mapa[i][j + 1].equals("S")) {
                            controlador.SuperIncrementarPuntos();
                        }
                        mapa[i][j + 1] += "P";
                        mapa[i][j] = mapa[i][j].replaceAll(" ", "C");
                        mapa[i][j] = mapa[i][j].replaceAll("S", "C");
                        mapa[i][j] = mapa[i][j].replaceAll("P", "");
                        direccion = "derecha";
                        posicionY = i;
                        posicionX = j + 1;
                        coordenadasPacman[0] = posicionY;
                        coordenadasPacman[1] = posicionX;
                        return;
                    } else if(mapa[i][j + 1].contains("X")) {
                        direccionDeseada = direccion;
                        direccionEnEspera = "derecha";
                    }
                    break;
                case "izquierda":
                    if(mapa[i][j - 1].contains(" ") || mapa[i][j - 1].contains("S") || mapa[i][j - 1].contains("C") || mapa[i][j - 1].contains("1")
                        || mapa[i][j - 1].contains("2") || mapa[i][j - 1].contains("3") || mapa[i][j - 1].contains("4")) {
                        controlador.verificarMuerte();
                        if(mapa[i][j - 1].equals(" ")) {
                            controlador.incrementarPuntos();
                        } else if(mapa[i][j - 1].equals("S")) {
                            controlador.SuperIncrementarPuntos();
                        }
                        mapa[i][j - 1] += "P";
                        mapa[i][j] = mapa[i][j].replaceAll(" ", "C");
                        mapa[i][j] = mapa[i][j].replaceAll("S", "C");
                        mapa[i][j] = mapa[i][j].replaceAll("P", "");
                        direccion = "izquierda";
                        posicionY = i;
                        posicionX = j - 1;
                        coordenadasPacman[0] = posicionY;
                        coordenadasPacman[1] = posicionX;
                        return;
                    } else if(mapa[i][j - 1].contains("X")) {
                        direccionDeseada = direccion;
                        direccionEnEspera = "izquierda";
                    }
                    break;
            }
        }
    }
}
```

Fantasma.java:

```java
import java.util.*;

public class Fantasma
{
    private Pacman pacman;
    private Mapa mapa;

    // Constructor que inicializa el mapa y el pacman
    public Fantasma(Mapa mapa, Pacman pacman) {
        this.mapa = mapa;
        this.pacman = pacman;
    }
/*
    Algortitmo pathfinder para mover los fantasmas, conocido como A* (A-star).
    Clase anidada que representa un nodo en el camino. Los nodos son las celdas en el mapa.
    Esta clase se utiliza para representar cada celda en el camino que el fantasma tomará para alcanzar a Pacman
    Cada nodo contiene su posición (x, y), los costos g, h y f, y una referencia a su nodo padre.
    El costo g representa la distancia desde el nodo inicial hasta el nodo actual.
    El costo h es una estimación de la distancia desde el nodo actual hasta el nodo objetivo.
    El costo f es la suma de g y h, y se utiliza para determinar el nodo con el menor costo total.
*/
private class Nodo {
    int x, y; // Coordenadas del nodo en el mapa
    int g, h, f; // Costos g, h y f. Los costos son la distancia de celdas que cuesta llegar al nodo en (x, y)
    Nodo padre; // Referencia al nodo padre

    // Constructor que inicializa las coordenadas, el nodo padre y los costos g y h
    Nodo(int x, int y, Nodo padre, int g, int h) {
        this.x = x; // Coordenada x del nodo actual
        this.y = y; // Coordenada y del nodo actual
        this.padre = padre; // Nodo padre (nodo anterior al actual)
        this.g = g; // Costo desde el nodo inicial hasta este nodo
        this.h = h; // Estimación del costo desde este nodo hasta el nodo objetivo
        this.f = g + h; // Costo total (g + h)
    }

    // Método equals para comparar dos nodos
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true; // Si los objetos son iguales, devuelve true
        if (obj == null || getClass() != obj.getClass()) return false; // Si el objeto es nulo o no es de la misma clase, devuelve false
        Nodo nodo = (Nodo) obj; // Convierte el objeto a Nodo
        return x == nodo.x && y == nodo.y; // Compara las coordenadas x e y
    }

    // Método hashCode para generar un código hash para el nodo
    @Override
    public int hashCode() {
        return Objects.hash(x, y);
    }
}
```

```java
// Método para encontrar el camino desde una posición inicial a una posición final
private List<Nodo> encontrarCamino(int startX, int startY, int endX, int endY) { // endX/endY son las coordenadas de Pacman, y startX/startY las del fantasma
    PriorityQueue<Nodo> abierta = new PriorityQueue<>(Comparator.comparingInt(n -> n.f)); // Cola de prioridad de nodos abiertos, ordenada por costo total f
    Set<Nodo> cerrada = new HashSet<>(); // Conjunto de nodos cerrados
    abierta.add(new Nodo(startX, startY, null, 0, Math.abs(startX - endX) + Math.abs(startY - endY))); // Añadir el nodo inicial a la cola de prioridad

    while (!abierta.isEmpty()) { // Mientras haya nodos en la cola de prioridad
        Nodo actual = abierta.poll(); // Obtener el nodo con el menor costo total f
        if (actual.x == endX && actual.y == endY) { // Si el nodo actual es el nodo objetivo
            List<Nodo> camino = new ArrayList<>(); // Crear una lista para almacenar el camino
            while (actual != null) { // Mientras el nodo actual no sea nulo
                camino.add(actual); // Añadir el nodo actual al camino
                actual = actual.padre; // Moverse al nodo padre
            }
            Collections.reverse(camino); // Invertir el camino para obtener el orden correcto
            return camino; // Devolver el camino
        }

        cerrada.add(actual); // Añadir el nodo actual al conjunto de nodos cerrados

        for (int[] dir : new int[][]{{0, -1}, {0, 1}, {-1, 0}, {1, 0}}) { // Para cada dirección posible (arriba, abajo, izquierda, derecha)
            int nuevoX = actual.x + dir[0]; // Calcular la nueva coordenada x
            int nuevoY = actual.y + dir[1]; // Calcular la nueva coordenada y
            if (nuevoX < 0 || nuevoY < 0 || nuevoX >= mapa.getArrayMapa1().length || nuevoY >= mapa.getArrayMapa1()[0].length) { // Si fuera del mapa
                continue; // Saltar a la siguiente iteración
            }
            if (mapa.getArrayMapa1()[nuevoX][nuevoY].equals("X")) { // Si la nueva coordenada es un obstáculo
                continue; // Saltar a la siguiente iteración
            }
            Nodo vecino = new Nodo(nuevoX, nuevoY, actual, actual.g + 1, Math.abs(nuevoX - endX) + Math.abs(nuevoY - endY)); // Crear un nuevo nodo vecino
            if (cerrada.contains(vecino)) { // Si el nodo vecino ya está en el conjunto de nodos cerrados
                continue; // Saltar a la siguiente iteración
            }
            abierta.add(vecino); // Añadir el nodo vecino a la cola de prioridad
        }
    }
    return null; // Si no se encuentra un camino, devolver null
}
```

```java
// Método para mover un fantasma en una dirección específica
private void moverFantasma(int fantasma, String idFantasma) {
    String[][] arrayMapa = mapa.getArrayMapa1();
    for (int i = 0; i < arrayMapa.length; i++) {
        for (int j = 0; j < arrayMapa[i].length; j++) {
            if (arrayMapa[i][j].contains(idFantasma)) {
                List<Nodo> camino = encontrarCamino(i, j, pacman.coordenadasPacman[0], pacman.coordenadasPacman[1]);
                if (camino != null && camino.size() > 1) {
                    Nodo siguientePaso = camino.get(1);
                    arrayMapa[siguientePaso.x][siguientePaso.y] += idFantasma;
                    arrayMapa[i][j] = arrayMapa[i][j].replaceAll(idFantasma, "");
                } else {
                    int[][] direcciones = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};
                    Collections.shuffle(Arrays.asList(direcciones));
                    for (int[] dir : direcciones) {
                        int nuevoX = i + dir[0];
                        int nuevoY = j + dir[1];
                        if (nuevoX >= 0 && nuevoY >= 0 && nuevoX < arrayMapa.length && nuevoY < arrayMapa[0].length && !arrayMapa[nuevoX][nuevoY].contains("X")) {
                            arrayMapa[nuevoX][nuevoY] += idFantasma;
                            arrayMapa[i][j] = arrayMapa[i][j].replaceAll(idFantasma, "");
                            break;
                        }
                    }
                }
                return;
            }
        }
    }
}
```

```java
    // Método para mover el fantasma 1
    public void moverFantasma1() {
        moverFantasma(1, "1");
    }

    // Método para mover el fantasma 2
    public void moverFantasma2() {
        moverFantasma(2, "2");
    }

    // Método para mover el fantasma 3
    public void moverFantasma3() {
        moverFantasma(3, "3");
    }

    // Método para mover el fantasma 4
    public void moverFantasma4() {
        moverFantasma(4, "4");
    }
```

Controlador.java:

```java
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Controlador implements KeyListener
{
    private final Mapa mapa;
    private final Ventana ventana;
    public static boolean juegoIniciado;
    private final Pacman pacman;
    private int puntos;
    private final Fantasma fantasmas;
    private int vidasRestantes = 3;

    String[][] arrayMapa;

    public Controlador() {
        mapa = new Mapa();
        ventana = new Ventana(this);
        pacman = new Pacman(this);
        fantasmas = new Fantasma(mapa, pacman);
        puntos = 0;
        arrayMapa = mapa.getArrayMapa1();
        ventana.addKeyListener(this);
        ventana.setFocusable(true);
        ventana.requestFocusInWindow();
    }

    public void iniciarJuego() {
        juegoIniciado = true;
        ventana.renderizar();
        new Thread(() -> {
            try {
                if(juegoIniciado) { new Thread(this::iniciarFantasma1).start(); }
                Thread.sleep(3500);
                if(juegoIniciado) { new Thread(this::iniciarFantasma2).start(); }
                Thread.sleep(3500);
                if(juegoIniciado) { new Thread(this::iniciarFantasma3).start(); }
                Thread.sleep(3500);
                if(juegoIniciado) { new Thread(this::iniciarFantasma4).start(); }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();

        while(juegoIniciado) {
            pacman.mover(Pacman.direccionDeseada);
            verificarMuerte();
            reaparecerComida();
            ventana.renderizar();
            try {
                Thread.sleep(120);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```java
public void iniciarFantasma1() {
    while(juegoIniciado) {
        fantasmas.moverFantasma1();
        ventana.renderizar();
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public void iniciarFantasma2() {
    while(juegoIniciado) {
        fantasmas.moverFantasma2();
        ventana.renderizar();
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
public void iniciarFantasma3() {
    while(juegoIniciado) {
        fantasmas.moverFantasma3();
        ventana.renderizar();
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public void iniciarFantasma4() {
    while(juegoIniciado) {
        fantasmas.moverFantasma4();
        ventana.renderizar();
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public Mapa getMapa() {
    return mapa;
}

public void incrementarPuntos() {
    puntos += 10;
}

public void SuperIncrementarPuntos() {
    puntos += 150;
}

public int getPuntos() {
    return puntos;
}

public void verificarMuerte() {
    for(int i = 0; i < arrayMapa.length; i++) {
        for(int j = 0; j < arrayMapa[i].length; j++) {
            if(arrayMapa[i][j].contains("P") &&
               (arrayMapa[i][j].contains("1") ||
                arrayMapa[i][j].contains("2") ||
                arrayMapa[i][j].contains("3") ||
                arrayMapa[i][j].contains("4"))) {
                muertePacman();
            }
        }
    }
}
```

```java
public void muertePacman() {
    vidasRestantes--;
    if(vidasRestantes <= 0) {
        try {
            Thread.sleep(121);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        juegoIniciado = false;
        System.out.println("Vidas restantes: " + vidasRestantes + ". GAME OVER");
        ventana.renderizar();
    } else if(vidasRestantes > 0){
        try {
            Thread.sleep(120); // Esperar al último movimiento antes de morir
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        juegoIniciado = false;
        System.out.println("Pacman ha muerto. Vidas restantes: " + vidasRestantes);
        reaparecerPacman();
        reaparecerFantasmas();
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        reaparecerPacman();
        juegoIniciado = true;
    }
}
```

```java
private void reaparecerPacman() { // Reestablecer la posición de Pacman
    for(int i = 0; i < arrayMapa.length; i++) {
        for(int j = 0; j < arrayMapa[i].length; j++) {
            if(arrayMapa[i][j].contains("P")) {
                arrayMapa[i][j] = arrayMapa[i][j].replace("P", "");
            }
        }
    }
    arrayMapa[13][15] = "P";
    pacman.coordenadasPacman[0] = 13;
    pacman.coordenadasPacman[1] = 15;

}
```

```java
public void reaparecerFantasmas() { // Restablecer las posiciones de los fantasmas
    for(int i = 0; i < arrayMapa.length; i++) {
        for(int j = 0; j < arrayMapa[i].length; j++) {
            if(arrayMapa[i][j].contains("1")) {
                arrayMapa[i][j] = arrayMapa[i][j].replace("1", "");
            }
            if(arrayMapa[i][j].contains("2")) {
                arrayMapa[i][j] = arrayMapa[i][j].replace("2", "");
            }
            if(arrayMapa[i][j].contains("3")) {
                arrayMapa[i][j] = arrayMapa[i][j].replace("3", "");
            }
            if(arrayMapa[i][j].contains("4")) {
                arrayMapa[i][j] = arrayMapa[i][j].replace("4", "");
            }
        }
    }
    arrayMapa[7][14] = "1";
    arrayMapa[7][16] = "2";
    arrayMapa[9][14] = "3";
    arrayMapa[9][16] = "4";

    new Thread(() -> {
        try {
            Thread.sleep(2500);
            if(juegoIniciado) { new Thread(this::iniciarFantasma1).start(); }
            Thread.sleep(3500);
            if(juegoIniciado) { new Thread(this::iniciarFantasma2).start(); }
            Thread.sleep(3500);
            if(juegoIniciado) { new Thread(this::iniciarFantasma3).start(); }
            Thread.sleep(3500);
            if(juegoIniciado) { new Thread(this::iniciarFantasma4).start(); }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }).start();
}
```

```java
public void reaparecerComida() {
    boolean hayComida = false;
    for(int i = 0; i < arrayMapa.length; i++) {
        for(int j = 0; j < arrayMapa[i].length; j++) {
            if(arrayMapa[i][j].contains(" ") || arrayMapa[i][j].contains("S")) {
                hayComida = true;
                break;
            }
        }
        if(hayComida) { break; }
    }

    if (!hayComida) {
        for (int i = 0; i < arrayMapa.length; i++) {
            for (int j = 0; j < arrayMapa[i].length; j++) {
                arrayMapa[i][j] = arrayMapa[i][j].replaceAll("C", " ");
            }
        }
    }
}
```

```java
public int getVidasRestantes() {
    return vidasRestantes;
}

@Override
public void keyPressed(KeyEvent e) {
    int keyCode = e.getKeyCode();
    switch(keyCode) {
        case KeyEvent.VK_UP:
            Pacman.direccionDeseada = "arriba";
            break;
        case KeyEvent.VK_DOWN:
            Pacman.direccionDeseada = "abajo";
            break;
        case KeyEvent.VK_RIGHT:
            Pacman.direccionDeseada = "derecha";
            break;
        case KeyEvent.VK_LEFT:
            Pacman.direccionDeseada = "izquierda";
            break;
    }
}

@Override
public void keyTyped(KeyEvent e) {}

@Override
public void keyReleased(KeyEvent e) {}
```

Ventana.java:

```java
import java.awt.*;
import java.awt.image.BufferStrategy;
import javax.swing.*;

public class Ventana extends JFrame {

    private Controlador controlador;
    private static final int tamañoCelda = 35;
    private static final int espacioExtraY = 7;

    // Imágenes
    public static Image pacmanArriba;
    public static Image pacmanAbajo;
    public static Image pacmanIzquierda;
    public static Image pacmanDerecha;
    public static Image pacmanDireccion;

    public Image fantasma1;
    public Image fantasma2;
    public Image fantasma3;
    public Image fantasma4;

    private String[][] mapa;

    public Ventana(Controlador controlador) {
        this.controlador = controlador;
        setTitle("PacMan SMA");
        setSize(1200, 800);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        setResizable(false);
        createBufferStrategy(2);
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        renderizar();
    }
```

```java
public void renderizar() {
    pacmanArriba = new ImageIcon(getClass().getResource("Imágenes/pacmanArriba.png")).getImage();
    pacmanAbajo = new ImageIcon(getClass().getResource("Imágenes/pacmanAbajo.png")).getImage();
    pacmanIzquierda = new ImageIcon(getClass().getResource("Imágenes/pacmanIzquierda.png")).getImage();
    pacmanDerecha = new ImageIcon(getClass().getResource("Imágenes/pacmanDerecha.png")).getImage();

    fantasma1 = new ImageIcon(getClass().getResource("Imágenes/fantasma1.png")).getImage();
    fantasma2 = new ImageIcon(getClass().getResource("Imágenes/fantasma2.png")).getImage();
    fantasma3 = new ImageIcon(getClass().getResource("Imágenes/fantasma3.png")).getImage();
    fantasma4 = new ImageIcon(getClass().getResource("Imágenes/fantasma4.png")).getImage();

    BufferStrategy bufferStrategy = getBufferStrategy();
    if (bufferStrategy == null) {
        createBufferStrategy(2);
        return;
    }

    Graphics g = bufferStrategy.getDrawGraphics();

    if (g != null) {
        g.setColor(Color.BLACK);
        g.fillRect(0, 0, getWidth(), getHeight());

        if (controlador.getVidasRestantes() > 0) {
            mapa = controlador.getMapa().getArrayMapa1();
        } else if (controlador.getVidasRestantes() <= 0) {
            mapa = controlador.getMapa().getArrayMapaGameOver();
        }
        int offsetX = (getWidth() - mapa[0].length * tamañoCelda) / 2;
        int offsetY = (getHeight() - mapa.length * tamañoCelda) / 2 + espacioExtraY;

        // RENDERIZAR PAREDES
        for (int i = 0; i < mapa.length; i++) {
            for (int j = 0; j < mapa[i].length; j++) {
                if (mapa[i][j].equals("X")) {
                    if (controlador.getVidasRestantes() > 0) {
                        g.setColor(Color.BLUE);
                    } else if (controlador.getVidasRestantes() <= 0) {
                        g.setColor(Color.RED);
                    }
                    g.fillRect(j * tamañoCelda + offsetX, i * tamañoCelda + offsetY, tamañoCelda, tamañoCelda);
                }
            }
        }
        // RENDERIZAR COMIDA
        for (int i = 0; i < mapa.length; i++) {
            for (int j = 0; j < mapa[i].length; j++) {
                if (mapa[i][j].contains(" ")) {
                    g.setColor(Color.GREEN);
                    int centroX = j * tamañoCelda + offsetX + tamañoCelda / 2;
                    int centroY = i * tamañoCelda + offsetY + tamañoCelda / 2;
                    int radioComida = tamañoCelda / 4;
                    g.fillOval(centroX - radioComida, centroY - radioComida, radioComida * 2, radioComida * 2);
                }
            }
        }

        // RENDERIZAR SUPERCOMIDA
        for (int i = 0; i < mapa.length; i++) {
            for (int j = 0; j < mapa[i].length; j++) {
                if (mapa[i][j].contains("S")) {
                    g.setColor(Color.WHITE);
                    int centroX = j * tamañoCelda + offsetX + tamañoCelda / 2;
                    int centroY = i * tamañoCelda + offsetY + tamañoCelda / 2;
                    int radioComida = tamañoCelda / 3;
                    g.fillOval(centroX - radioComida, centroY - radioComida, radioComida * 2, radioComida * 2);
                }
            }
        }
```

```
        // RENDERIZAR PACMAN
        for (int i = 0; i < mapa.length; i++) {
            for (int j = 0; j < mapa[i].length; j++) {
                if (mapa[i][j].contains("P")) {
                    int centroX = j * tamañoCelda + offsetX + tamañoCelda / 2;
                    int centroY = i * tamañoCelda + offsetY + tamañoCelda / 2;
                    int radioPacman = tamañoCelda / 2;

                    String direccion = Pacman.direccion;
                    if (direccion != null) {
                        switch (direccion) {
                            case "arriba":
                                pacmanDireccion = pacmanArriba;
                                break;
                            case "abajo":
                                pacmanDireccion = pacmanAbajo;
                                break;
                            case "izquierda":
                                pacmanDireccion = pacmanIzquierda;
                                break;
                            case "derecha":
                                pacmanDireccion = pacmanDerecha;
                                break;
                        }
                    } else {
                        pacmanDireccion = pacmanDerecha;
                    }

                    g.drawImage(pacmanDireccion, centroX - radioPacman, centroY - radioPacman, radioPacman * 2, radioPacman * 2, null);
                }
            }
        }
```

```
            // RENDERIZAR VIDAS
            g.setColor(Color.WHITE);
            g.setFont(new Font("Verdana", Font.BOLD, 25));

            if (controlador.getVidasRestantes() >= 0) {
                g.drawString("Vidas restantes: " + controlador.getVidasRestantes(), 890, 90);
            } else {
                g.drawString("Vidas restantes: 0", 890, 90);
            }

            g.dispose();
            bufferStrategy.show();
        }
    }
```

```
        // RENDERIZAR FANTASMAS
        for (int i = 0; i < mapa.length; i++) {
            for (int j = 0; j < mapa[i].length; j++) {
                if (mapa[i][j].contains("1")) {
                    g.drawImage(fantasma1, j * tamañoCelda + offsetX, i * tamañoCelda + offsetY, tamañoCelda, tamañoCelda, null);
                }
                if (mapa[i][j].contains("2")) {
                    g.drawImage(fantasma2, j * tamañoCelda + offsetX, i * tamañoCelda + offsetY, tamañoCelda, tamañoCelda, null);
                }
                if (mapa[i][j].contains("3")) {
                    g.drawImage(fantasma3, j * tamañoCelda + offsetX, i * tamañoCelda + offsetY, tamañoCelda, tamañoCelda, null);
                }
                if (mapa[i][j].contains("4")) {
                    g.drawImage(fantasma4, j * tamañoCelda + offsetX, i * tamañoCelda + offsetY, tamañoCelda, tamañoCelda, null);
                }
            }
        }

        // RENDERIZAR PUNTOS
        g.setColor(Color.WHITE);
        g.setFont(new Font("Verdana", Font.BOLD, 25));
        g.drawString("PUNTOS: " + controlador.getPuntos(), 57, 90);
```

```
        // RENDERIZAR VIDAS
        g.setColor(Color.WHITE);
        g.setFont(new Font("Verdana", Font.BOLD, 25));

        if (controlador.getVidasRestantes() >= 0) {
            g.drawString("Vidas restantes: " + controlador.getVidasRestantes(), 890, 90);
        } else {
            g.drawString("Vidas restantes: 0", 890, 90);
        }

        g.dispose();
        bufferStrategy.show();
    }
}
```