UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE SEDE SANTO DOMINGO
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS
CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	Abril 2023 – Octubre 2023
ASIGNATURA	:	POO
TEMA	:	Tarea 4 Juego
NOMBRES	:	Eduardo Ordoñez Kelvin Quezada
NIVEL-PARALELO	:	Segundo “A”
DOCENTE	:	Ing. Cevallos Farias Javier Moyota
FECHA DE ENTREGA	:	14/08/2023

SANTO DOMINGO - ECUADOR

2023

Guía de Juego Pacman

Paso 1

En nuestra clase Pacman colocamos extends ya que permite heredar todas las características y funcionalidades de una ventana de la interfaz creamos también un constructor llamado Pacman para llamar llamar nuestra interfaces y creamos otro método llamado initUI para inicializar y configurar la interfaz gráfica de la ventana de juego

Este es el método principal main del programa. Se ejecuta cuando se inicia la aplicación. Utiliza EventQueue.invokeLater() para crear y mostrar una instancia de la clase Pacman. En resumen, crea una ventana del juego y la hace visible.

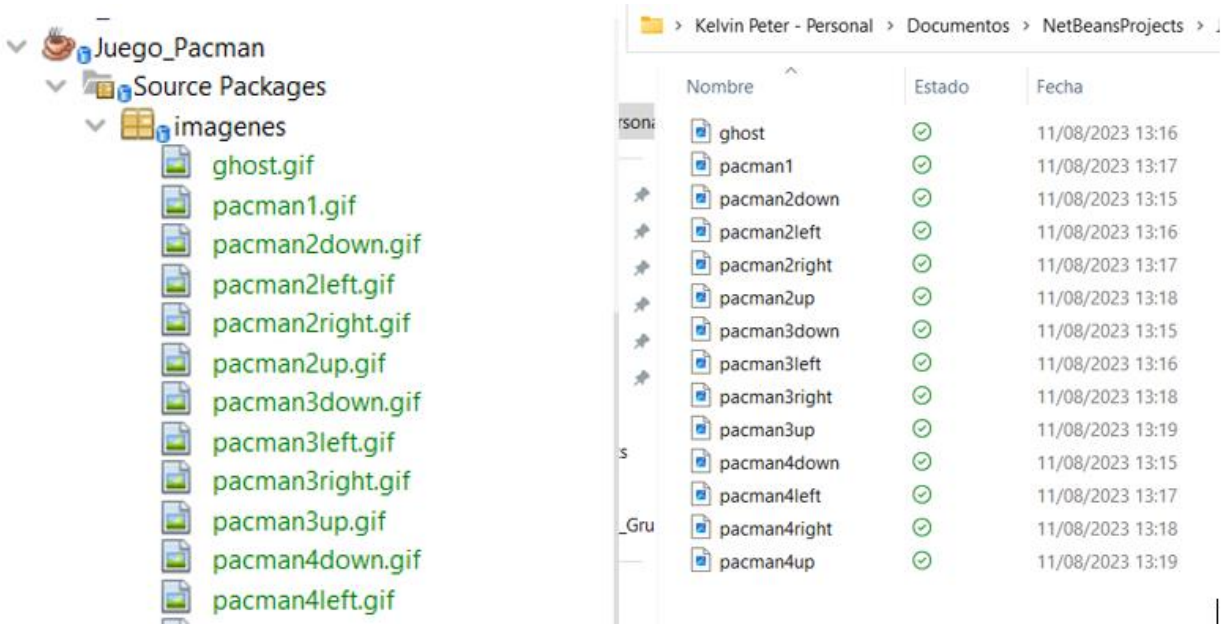
```
public class Pacman extends JFrame {
    public Pacman() {
        // Inicializa la interfaz gráfica del juego
        initUI();
    }
    private void initUI() {
        // Agrega un tablero (instancia de la clase 'tablero') a la ventana
        add(new tablero());
        // Configura las propiedades de la ventana
        setTitle(title: "Juego de Pac-Man"); // Título de la ventana
        setDefaultCloseOperation(operation:EXIT_ON_CLOSE);     // Termina la aplicación al cerrar la ventana
        setSize(width: 380, height:420);                        // Tamaño de la ventana
        setLocationRelativeTo(c: null);                // Centra la ventana en la pantalla
        setVisible(b: true);                           // Hace la ventana visible
    }
    public static void main(String[] args) {
        // Ejecuta la creación y visualización de la ventana en el hilo de eventos
        EventQueue.invokeLater(() -> {
            Pacman ex = new Pacman(); // Crea una instancia de la clase 'Pacman'
            ex.setVisible(b: true);      // Hace la ventana visible
        });
    }
}
```

Resumen: *Esta parte del código se encarga de crear la ventana principal del juego Pac-Man y configurar su interfaz gráfica. La clase Pacman actúa como la entrada principal del programa y se encarga de iniciar y gestionar la visualización del juego.*

Paso 2

Guía de Juego Pacman

Creamos un paquete para subir nuestras imágenes para después llamar cada imagen para el juego lo colocamos en nuestro ejercicio del programa.



Resumen: esta parte es muy importante para subir nuestras imágenes a nuestra carpeta del programa para después llamar cada una de las imágenes en nuestro juego.

Paso 3

Este código define una clase llamada tablero, que extiende la clase JPanel y se implementa como un ActionListener. La clase contiene numerosas variables de instancia que representan diversos aspectos del juego, como el tamaño del tablero, la fuente de texto, las imágenes de los personajes, la posición y el movimiento de Pacman, la matriz de diseño del laberinto, y más. Estas variables se utilizan para controlar y representar el estado del juego y sus elementos.

```
public class tablero extends JPanel implements ActionListener {
// Declare instance variables
    private Dimension d; // Almacena el tamaño del tablero
    private final Font smallfont = new Font(name: "Helvetica", style: Font.BOLD, size: 14); // Fuente para el texto en el juego
    private Image ii; // Almacena una imagen
    private final Color dotcolor = new Color(r: 192, g: 192, b: 0); // Color de los puntos en el juego
    private Color mazecolor; // Color del laberinto
    private boolean ingame = false; // Indica si el juego está en curso o no
    private boolean dying = false; // Indica si el jugador está perdiendo una vida
    private final int blocksize = 24; //Tamaño de cada bloque en el laberinto.
    private final int nrofblocks = 15; //Número de bloques en cada fila/columna
    private final int scrsize = nrofblocks * blocksize; // Tamaño total de la pantalla basado en el tamaño del bloque y el número de bloques
    private final int pacanimdelay = 2;// Retraso de animación para Pacman
    private final int pacmananimcount = 4;// Número de fotogramas en la animación de Pacman.
    private final int maxghosts = 12;// Número máximo de fantasmas en el juego.
    private final int pacmanspeed = 6;// Velocidad de movimiento de Pacman.
    // Variables de animación
    private int pacanimcount = pacanimdelay;
    private int pacanimdir = 1;
    private int pacmananimpos = 0;
    private int nrofghosts = 6;// Número actual de fantasmas.
    private int pacsleft, score;// Vidas restantes y puntuación.
    private int[] dx, dy;// Matrices para el movimiento de fantasmas.
    // Matrices para atributos fantasma
    private int[] ghostx, ghosty, ghostdx, ghostdy, ghostspeed;
// Imágenes para elementos del juego.
    private Image ghost;
    private Image pacman1, pacman2up, pacman2left, pacman2right, pacman2down;
    private Image pacman3up, pacman3down, pacman3left, pacman3right;
    private Image pacman4up, pacman4down, pacman4left, pacman4right;
// Posición y movimiento de Pacman
    private int pacmanx, pacmany, pacmandx, pacmandy;
    private int reqdx, reqdy, viewdx, viewdy;
```

Resumen: *daremos a conocer los métodos y eventos de nuestro fragmento del código de nuestro juego que nos ayuda mucho para nuestro juego.*

Paso 4

Daremos a conocer el diseño del laberinto Este bloque de código define la matriz leveldata, que representa el diseño del laberinto del juego.

```
// Datos de diseño del laberinto
    private final short leveldata[] = {
        19, 26, 26, 26, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,
        21, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
        21, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
        21, 0, 0, 0, 17, 16, 16, 24, 16, 16, 16, 16, 16, 16, 20,
        17, 18, 18, 18, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 20,
        17, 16, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 16, 24, 20,
        25, 16, 16, 16, 24, 24, 28, 0, 25, 24, 24, 16, 20, 0, 21,
        1, 17, 16, 20, 0, 0, 0, 0, 0, 0, 0, 17, 20, 0, 21,
        1, 17, 16, 16, 18, 18, 22, 0, 19, 18, 18, 16, 20, 0, 21,
        1, 17, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 20, 0, 21,
        1, 17, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 20, 0, 21,
        1, 17, 16, 16, 16, 16, 16, 18, 16, 16, 16, 16, 20, 0, 21,
        1, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0, 21,
        1, 25, 24, 24, 24, 24, 24, 24, 24, 24, 16, 16, 16, 18, 20,
        9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 25, 24, 24, 24, 28
    };
```

Resumen: *Cada número en la matriz corresponde a un tipo de celda en el laberinto: el 19 Celda superior izquierda del laberinto, 26 Pared horizontal, 18 Pared vertical, 22 Celda superior derecha del laberinto, 21 Celda inferior izquierda del laberinto, 20 Celda inferior derecha del laberinto, 17 Espacio vacío, 16 Punto en el laberinto, 24 Punto grande (poder) en el laberinto, 25 Túnel izquierdo del laberinto, 28 Túnel derecho del laberinto, 8 Celda con portal (utilizado para teleportar fantasmas).*

Paso 5

En este fragmento de código, se definen las velocidades de movimiento válidas para Pacman (validspeeds), junto con la velocidad máxima permitida (maxspeed) y la velocidad actual de Pacman (currentspeed).

```
// Velocidades de movimiento válidas para Pacman
    private final int validspeeds[] = {1, 2, 3, 4, 6, 8};
    private final int maxspeed = 6;
// Velocidad de movimiento actual de Pacman.
    private int currentspeed = 3;
```

Resumen: *Dará a conocer la velocidad de nuestro movimiento del pacman y nos dara a conocer la velocidad actual del Pacman.*

Paso 6

Estas líneas declaran la matriz screendata, que se utiliza para almacenar datos de diseño de la pantalla del juego, como las ubicaciones de los puntos y los puntos grandes. También se declara el temporizador timer, que se utiliza para el bucle principal del juego y para controlar la actualización del juego a intervalos regulares.

```
// Matriz para almacenar datos de diseño de laberinto.
private short[] screendata;
// Temporizador para bucle de juego.
private Timer timer;
```

Resumen: Se dará a conocer una matriz para almacenar los datos del diseño y dar a conocer el temporizador para el juego.

Paso 7

Este constructor se encarga de preparar el panel de juego para la interacción del jugador, cargando imágenes, inicializando variables y configurando la captura de eventos de teclado para el control del juego

```
// Constructor de la clase "tablero".
public tablero() {
    // Inicializar el juego.
    // Cargar las imágenes necesarias para el juego.
    loadImages();
    // Inicializar las variables necesarias para el juego.
    initVariables();
    // Configurar el oyente clave para la entrada del usuario.
    addKeyListener(new TAdapter());
    // Hacer que el panel sea foco para que pueda recibir eventos de teclado.
    setFocusable(focusable:true);
    // Establecer el color de fondo del panel a negro.
    setBackground(bg: Color.BLACK);
    // Habilitar el doble almacenamiento en búfer para lograr una representación fluida.
    setDoubleBuffered(aFlag: true);
}
```

Resumen: El constructor llamado tablero se ejecuta cuando se crea una instancia de la clase. Aquí se realiza la inicialización necesaria para configurar el panel de juego y prepararlo para la interacción del jugador.

Paso 8

Este método se encarga de inicializar las variables y configuraciones necesarias para el funcionamiento del juego. ya que inicializa el arreglo establece el color del laberinto se crea una instancia de la clase Dimensión para así definir el tamaño del juego y inicializa el arreglo para almacenar la información también se crea otra instancia para crear el objeto Timer ya que establece bucle del juego para invocar el método actionPerformen cada 40 milisegundos. El método addNotify anuala si de la clase Component que se llama cuando el componente del panel del juego se agrega un comentario. llamamos al método initGame para realizar las configuraciones iniciales del juego.

```
    private void initVariables() {
// Inicializar el arreglo que almacena la información de la pantalla (laberinto)
        screendata = new short[nrofblocks * nrofblocks];
        // Establecer el color del laberinto
        mazecolor = new Color(r: 5, g: 100, b: 5);
        // Crear una instancia de Dimension para el tamaño de la pantalla
        d = new Dimension(width: 400, height:400);
        // Inicializar arreglos para los fantasmas
        ghostx = new int[maxghosts];
        ghostdx = new int[maxghosts];
        ghosty = new int[maxghosts];
        ghostdy = new int[maxghosts];
        ghostspeed = new int[maxghosts];
        // Inicializar arreglos para los movimientos de los fantasmas y Pacman
        dx = new int[4];
        dy = new int[4];
// Inicializar y configurar el temporizador del bucle del juego
        timer = new Timer(delay: 40, listener: this);// Cada 40 milisegundos se invocará el actionPerformed()
        // Iniciar el temporizador
        timer.start();
    }
/// método addNotify() para inicializar el juego cuando se agrega a un contenedor
    @Override
    public void addNotify() {
        super.addNotify();
        initGame();
    }
```

Resumen: estos métodos se encargan de inicializar y configurar las variables y objetos necesarios para el funcionamiento del juego, tanto cuando se inicia el programa como cuando se agrega el panel del juego a un contenedor.

Paso 9

Este método doAnim es para crear la animación de la boca del Pacman utilizamos un decremento y utilizamos una condicional para para verificar si es momento de cambiar la animación en cero y se reiniciará el contador de la animación y también vamos a cambiar la posición de la animación del Pacman utilizamos otro if para comprobar la posición de la animación si está en el último cuadro o en el primero y por último cambiamos la dirección de la animación invirtiendo el signo de pacanimdir

```
    // Método para realizar la animación de la boca de Pacman
    private void doAnim() {
// Decrementar el contador de la animación
        pacanimcount--;
// Verificar si es momento de cambiar la animación
        if (pacanimcount <= 0) {
            // Reiniciar el contador de la animación
            pacanimcount = pacanimdelay;
            // Cambiar la posición de la animación de Pacman
            pacmananimpos = pacmananimpos + pacanimdir;
// Comprobar si la posición de la animación está en el último cuadro o en el primero
            if (pacmananimpos == (pacmananimcount - 1) || pacmananimpos == 0) {
                // Cambiar la dirección de la animación invirtiendo el signo de pacanimdir
                pacanimdir = -pacanimdir;
            }
        }
    }
```

Paso 10

Este método controla el flujo principal del juego. Dependiendo de si Pacman está muriendo o no, se realizan diferentes acciones como mover a Pacman, dibujar elementos en la pantalla, mover a los fantasmas y verificar el estado del laberinto.

```
// Método para ejecutar el ciclo principal del juego
    private void playGame(Graphics2D g2d) {
// Verificar si Pacman está en proceso de morir
        if (dying) {
// Realizar acciones relacionadas con la muerte de Pacman
            death();

        } else {
            // Mover a Pacman
            movePacman();
            // Dibujar a Pacman en la pantalla
            drawPacman(g2d);
            // Mover a los fantasmas
            moveGhosts(g2d);
            // Verificar el estado del laberinto
            checkMaze();
        }
    }
```

Paso 11

Este método se utiliza para dibujar la pantalla de introducción del juego, incluyendo un fondo coloreado, un rectángulo con bordes, y un mensaje de inicio en el centro de la pantalla.

```
// Método para mostrar la pantalla de introducción
    private void showIntroScreen(Graphics2D g2d) {
// Establecer el color de fondo de la pantalla de introducción
        g2d.setColor(new Color(r: 0, g: 32, b: 48));
        // Dibujar un rectángulo en el centro de la pantalla para el mensaje
        g2d.fillRect(x: 50, scrsize / 2 - 30, scrsize - 100, height:50);
        // Establecer el color de los bordes del rectángulo
        g2d.setColor(c: Color.white);
        // Dibujar el borde del rectángulo
        g2d.drawRect(x: 50, scrsize / 2 - 30, scrsize - 100, height:50);
        // Mensaje a mostrar en la pantalla de introducción
        String s = "Presiona s para empezar.";
        // Crear una fuente para el mensaje
        Font small = new Font(name: "Helvetica", style: Font.BOLD, size: 15);
        // Obtener información de la fuente para medir el ancho del mensaje
        FontMetrics metr = this.getFontMetrics(font: small);
        // Establecer el color del texto
        g2d.setColor(c: Color.white);
        // Establecer la fuente para el texto
        g2d.setFont(font: small);
        // Dibujar el mensaje centrado en la pantalla de introducción
        g2d.drawString(str:s, (scrsize - metr.stringWidth(str:s)) / 2, scrsize / 2);
    }
```

Paso 12

Este método drawScore se utiliza para dibujar la puntuación actual del juego y mostrar las imágenes de Pacman que representan las vidas restantes en la parte inferior de la pantalla del juego.

```
// Método para dibujar la puntuación y las vidas restantes de Pacman
    private void drawScore(Graphics2D g) {

        int i;
        String s;
        // Establecer la fuente y el color para la puntuación
        g.setFont(font: smallfont);
        g.setColor(new Color(r: 96, g: 128, b: 255));
        // Crear un mensaje de puntuación en forma de cadena
        s = "Score: " + score;
        // Dibujar la puntuación en la pantalla
        // scrsize / 2 + 96 es la posición horizontal donde se dibujará la puntuación
        // scrsize + 16 es la posición vertical donde se dibujará la puntuación
        g.drawString(str:s, scrsize / 2 + 96, scrsize + 16);
        // Dibujar las vidas restantes de Pacman en la parte inferior
        // Itera a través de las vidas restantes y dibuja las imágenes de Pacman
        for (i = 0; i < pacsleft; i++) {
            // Dibuja la imagen de Pacman (pacman3left) en la posición (i * 28 + 8, scrsize + 1)
            g.drawImage(img:pacman3left, i * 28 + 8, scrsize + 1, observer: this);
        }
    }
```

Paso 13

El método checkMaze() se utiliza para verificar si todo los puntos en el laberinto han sido comidos por Pacman, ya que controla la lógica para avanzar al siguiente nivel una vez que Pacman ha comido todos los puntos en el laberinto.

```
private void checkMaze() {
    short i = 0;
    boolean finished = true;
    while (i < nrofblocks * nrofblocks && finished) {// Recorre todos los bloques en el laberinto
        // Verifica si el bloque contiene puntos (bits 4 y 5 están establecidos)
        if ((screendata[i] & 48) != 0) {
            // Aún hay puntos sin comer, por lo que no hemos terminado
            finished = false;
        }
        i++;
    }
    if (finished) { // Si todos los puntos han sido comidos (finished es verdadero)
        // Incrementa la puntuación en 50 puntos
        score += 50;
        if (nrofghosts < maxghosts) {// Si el número actual de fantasmas es menor que el máximo permitido
            nrofghosts++;// Incrementa el número de fantasmas
        }
        // Si la velocidad actual del juego es menor que la velocidad máxima permitida
        if (currentspeed < maxspeed) {
            currentspeed++; // Incrementa la velocidad del juego
        }
        // Inicia un nuevo nivel reiniciando el laberinto y los elementos del juego
        initLevel();
    }
}
```

Paso 14

El método death() gestiona la muerte de Pacman, actualiza el número de vidas restantes y decide si el juego ha terminado o si se debe reiniciar para continuar con el mismo nivel.

```
private void death() {//murio
    // Reduce el número de vidas restantes
    pacsleft--;
    // Si el número de vidas restantes es igual a cero
    if (pacsleft == 0) {
        // Establece el estado del juego como "no en juego" (game over)
        ingame = false;
    }
    // Continúa con el mismo nivel reiniciando los elementos del juego
    continueLevel();
}
```

Paso 15

El método moveGhosts(Graphics2D g2d) se utiliza para controlar el movimiento y la interacción de los fantasmas en el juego. creamos un for (i = 0; i < nrofghosts; i++) ya que recorre todo los fantasmas utilizamos un if (ghostx[i] % blocksize == 0 && ghosty[i] % blocksize == 0) para la verificación si el fantasmas se encuentra en una posición de bloque, creamos un pos para calcular la posición actual del fantasma en la matriz screendata, utilizamos una condicional if para la verificar las direcciones disponibles para el movimiento del fantasma y para verificar si el fantasma está atrapado en la posición, escoge aleatoriamente una de las direcciones disponibles, utilizamos una actualización de la posición del fantasma de acuerdo a su dirección.

```
    private void moveGhosts(Graphics2D g2d) {//mover fantasmas
//creamo variables
        short i; int pos; int count;
        // Recorre todos los fantasmas
        for (i = 0; i < nrofghosts; i++) {
            // Verifica si el fantasma se encuentra en una posición de bloque completa
            if (ghostx[i] % blocksize == 0 && ghosty[i] % blocksize == 0) {
                // Calcula la posición actual del fantasma en la matriz screendata
                pos = ghostx[i] / blocksize + nrofblocks * (int) (ghosty[i] / blocksize);
                count = 0;
                // Verifica las direcciones disponibles para el movimiento del fantasma
                if ((screendata[pos] & 1) == 0 && ghostdx[i] != 1) {
                    dx[count] = -1;
                    dy[count] = 0;
                    count++;
                }
                if ((screendata[pos] & 2) == 0 && ghostdy[i] != 1) {
                    dx[count] = 0;
                    dy[count] = -1;
                    count++;
                }
                if ((screendata[pos] & 4) == 0 && ghostdx[i] != -1) {
                    dx[count] = 1;
                    dy[count] = 0;
                    count++;
                }
                if ((screendata[pos] & 8) == 0 && ghostdy[i] != -1) {
                    dx[count] = 0;
                    dy[count] = 1;
                    count++;
                }
```

```
                if (count == 0) {
                    if ((screendata[pos] & 15) == 15) {
                        ghostdx[i] = 0;
                        ghostdy[i] = 0;
                    } else {
                        ghostdx[i] = -ghostdx[i];
                        ghostdy[i] = -ghostdy[i];
                    }
                } else {
                    // Escoge aleatoriamente una de las direcciones disponibles
                    count = (int) (Math.random() * count);
                    if (count > 3) {
                        count = 3;
                    }
                    ghostdx[i] = dx[count];
                    ghostdy[i] = dy[count];
                }
            }
            // Actualiza la posición del fantasma de acuerdo a su dirección y velocidad
            ghostx[i] = ghostx[i] + (ghostdx[i] * ghostspeed[i]);
            ghosty[i] = ghosty[i] + (ghostdy[i] * ghostspeed[i]);
            // Dibuja el fantasma en su nueva posición
            drawGhost(g2d, ghostx[i] + 1, ghosty[i] + 1);
            // Verifica si Pacman colisiona con un fantasma
            if (pacmanx > (ghostx[i] - 12) && pacmanx < (ghostx[i] + 12)
                    && pacmany > (ghosty[i] - 12) && pacmany < (ghosty[i] + 12)
                    && ingame) {
                // Establece el estado de "muriendo" para Pacman
                dying = true;
            }
        }
    }
```

Paso 16

Creamos un método para realizar la sobreescritura para los métodos para dibujar en la posición de (x y y) del objeto.

```
//realiza sobreescritura de metodos para dibujar drawGhost
    private void drawGhost(Graphics2D g2d, int x, int y) {
// Dibuja la imagen del fantasma en la posición (x, y) utilizando el objeto Graphics2D
        g2d.drawImage(img:ghost, x, y, observer: this);
    }
```

Paso 17

El método movePacman() se encarga de controlar el movimiento del personaje principal, del Pacman creamos una condicional para solicitar la dirección opuesta, actualizacion sin cambiar la vista creamos contra condicional if para la verificación si el pacman está en una posición del

cuadrado también cuenta con la verificación de la posición donde no puede moverse daremos a conocer las actualizaciones de nuestro pacman.

```
    private void movePacman() {
        int pos;
        short ch;
// Si se solicita un cambio de dirección opuesto, actualiza la dirección sin cambiar la vista
        if (reqdx == -pacmandx && reqdy == -pacmandy) {
            pacmandx = reqdx;
            pacmandy = reqdy;
            viewdx = pacmandx;
            viewdy = pacmandy;
        }
        // Verifica si Pacman está en una posición de cuadrícula
        if (pacmanx % blocksize == 0 && pacmany % blocksize == 0) {
            pos = pacmanx / blocksize + nrofblocks * (int) (pacmany / blocksize);
            ch = screendata[pos];
            // Verifica si hay una píldora en la celda actual
            if ((ch & 16) != 0) {
                screendata[pos] = (short) (ch & 15);// Elimina la píldora de la celda
                score++;// Incrementa la puntuación del jugador
            }
            // Verifica si se solicita un cambio de dirección y si es posible moverse en esa dirección
            if (reqdx != 0 || reqdy != 0) {
                if (!((reqdx == -1 && reqdy == 0 && (ch & 1) != 0)
                        || (reqdx == 1 && reqdy == 0 && (ch & 4) != 0)
                        || (reqdx == 0 && reqdy == -1 && (ch & 2) != 0)
                        || (reqdx == 0 && reqdy == 1 && (ch & 8) != 0))) {
                    pacmandx = reqdx;
                    pacmandy = reqdy;
                    viewdx = pacmandx;
                    viewdy = pacmandy;
                }
            }
```

```
            // Verifica si Pacman está en una posición donde no puede moverse
            if ((pacmandx == -1 && pacmandy == 0 && (ch & 1) != 0)
                    || (pacmandx == 1 && pacmandy == 0 && (ch & 4) != 0)
                    || (pacmandx == 0 && pacmandy == -1 && (ch & 2) != 0)
                    || (pacmandx == 0 && pacmandy == 1 && (ch & 8) != 0)) {
                pacmandx = 0;
                pacmandy = 0;
            }
        }
        // Actualiza la posición de Pacman basado en la dirección y la velocidad
        pacmanx = pacmanx + pacmanspeed * pacmandx;
        pacmany = pacmany + pacmanspeed * pacmandy;
    }
```

Paso 18

El método drawPacman dará a conocer un objeto fr Graphics2Dcomo argumento ya que se utiliza para dibujar en el contexto gráfico que dira que si viewdx es igual a -1 significa que el pacman esta mirando hacia la izquierda, si el viewdx es igual a 1 el pacman mirara hacia la derecha,

si el viewdx es igual a -1 el pacman mira hacia arriba, y si no se cumple las condiciones el pacman mira hacia abajo.

```
private void drawPacman(Graphics2D g2d) {
    if (viewdx == -1) {
        // Si viewdx es igual a -1, significa que Pacman está mirando hacia la izquierda
        // Dibuja Pacman mirando hacia la izquierda
        drawPacnanLeft(g2d);
    } else if (viewdx == 1) {
        // Si viewdx es igual a 1, significa que Pacman está mirando hacia la derecha
        // Dibuja Pacman mirando hacia la derecha
        drawPacmanRight(g2d);
    } else if (viewdy == -1) {
        // Si viewdy es igual a -1, significa que Pacman está mirando hacia arriba
        // Dibuja Pacman mirando hacia arriba
        drawPacmanUp(g2d);
    } else {
        // Si no se cumple ninguna de las condiciones anteriores, significa que Pacman está mirando hacia abajo
        // Dibuja Pacman mirando hacia abajo
        drawPacmanDown(g2d);
    }
}
```

Paso 19

Se dará a conocer una sobreescritura donde seleccionamos la imagen del pacman ya que se basan en dibujar la imagen en la segunda, tercera, cuarta del pacman hacia arriba ya que si ninguna de las condiciones se cumplen la imagen se predetermina en el pacman.

```
//aplico la sobreescritura del metodo drawPacman
    private void drawPacmanUp(Graphics2D g2d) {
// Seleccionar la imagen de Pacman a dibujar basada en el valor de pacmananimpos
        switch (pacmananimpos) {
            case 1:
                // Dibujar la segunda imagen de Pacman mirando hacia arriba
                g2d.drawImage(img: pacman2up, pacmanx + 1, pacmany + 1, observer: this);

                break;
            case 2:
                // Dibujar la tercera imagen de Pacman mirando hacia arriba
                g2d.drawImage(img: pacman3up, pacmanx + 1, pacmany + 1, observer: this);
                break;
            case 3:
                // Dibujar la cuarta imagen de Pacman mirando hacia arriba
                g2d.drawImage(img: pacman4up, pacmanx + 1, pacmany + 1, observer: this);
                break;
            default:
                //Si no coincide con ninguna de las opciones anteriores, dibujar la imagen predeterminada de Pacman
                g2d.drawImage(img: pacman1, pacmanx + 1, pacmany + 1, observer: this);
                break;
        }
    }
```

Paso 20

Este método drawPacmanDown se dará a conocer el valor de pacmananimpos determina la animación actual de Pacman, si pacmananimpos es 1, dibuja la segunda imagen de Pacman

mirando hacia abajo, Si pacmananimpos es 2, dibuja la tercera imagen de Pacman mirando hacia abajo, Si pacmananimpos es 3, dibuja la cuarta imagen de Pacman mirando hacia abajo, Si pacmananimpos no es ninguno de los casos anteriores, dibuja la primera imagen de Pacman.

```
//aplico la sobreescritura del metodo drawPacman
private void drawPacmanDown(Graphics2D g2d) {
    // El valor de pacmananimpos determina la animación actual de Pacman
    switch (pacmananimpos) {
        case 1:
            // Si pacmananimpos es 1, dibuja la segunda imagen de Pacman mirando hacia abajo
            g2d.drawImage(img: pacman2down, pacmanx + 1, pacmany + 1, observer: this);
            break;
        case 2:
            // Si pacmananimpos es 2, dibuja la tercera imagen de Pacman mirando hacia abajo
            g2d.drawImage(img: pacman3down, pacmanx + 1, pacmany + 1, observer: this);
            break;
        case 3:
            // Si pacmananimpos es 3, dibuja la cuarta imagen de Pacman mirando hacia abajo
            g2d.drawImage(img: pacman4down, pacmanx + 1, pacmany + 1, observer: this);
            break;
        default:
            // Si pacmananimpos no es ninguno de los casos anteriores, dibuja la primera imagen de Pacman
            g2d.drawImage(img: pacman1, pacmanx + 1, pacmany + 1, observer: this);
            break;
    }
}
```

Paso 21

Este método se encarga de dibujar la imagen de Pacman cuando está mirando hacia la izquierda utilizamos un switch para seleccionar cada opciones la primera opción es para dibujar la segunda imagen del Pacman mirando hacia arriba, Si pacmananimpos es 2, dibujar la tercera imagen de Pacman mirando hacia la izquierda, Si pacmananimpos es 3, dibujar la cuarta imagen de Pacman mirando hacia la izquierda, Si no coincide con ninguna de las opciones anteriores, dibujar la imagen predeterminada de Pacman.

```
    private void drawPacnanLeft(Graphics2D g2d) {
// Seleccionar la imagen de Pacman a dibujar basada en el valor de pacmananimpos
        switch (pacmananimpos) {
            case 1:
                // Dibujar la segunda imagen de Pacman mirando hacia abajo
                g2d.drawImage(img:pacman2left, pacmanx + 1, pacmany + 1, observer: this);
                break;
            case 2:
                // Dibujar la tercera imagen de Pacman mirando hacia abajo
                g2d.drawImage(img:pacman3left, pacmanx + 1, pacmany + 1, observer: this);
                break;
            case 3:
                // Dibujar la cuarta imagen de Pacman mirando hacia abajo
                g2d.drawImage(img:pacman4left, pacmanx + 1, pacmany + 1, observer: this);
                break;
            default:
                // Si no coincide con ninguna de las opciones anteriores, dibujar la imagen predeterminada de Pacman
                g2d.drawImage(img:pacman1, pacmanx + 1, pacmany + 1, observer: this);
                break;
        }
    }
```

Paso 22

Este método maneja la animación de Pacman cuando está mirando hacia la derecha, cambiando entre diferentes imágenes para crear la ilusión de movimiento.

```
    private void drawPacmanRight(Graphics2D g2d) {
// Seleccionar la imagen de Pacman a dibujar basada en el valor de pacmananimpos
        switch (pacmananimpos) {
            case 1:
                // Dibujar la segunda imagen de Pacman mirando hacia la derecha
                g2d.drawImage(img:pacman2right, pacmanx + 1, pacmany + 1, observer: this);
                break;
            case 2:
                 // Dibujar la tercera imagen de Pacman mirando hacia la derecha
                g2d.drawImage(img:pacman3right, pacmanx + 1, pacmany + 1, observer: this);
                break;
            case 3:
                // Dibujar la cuarta imagen de Pacman mirando hacia la derecha
                g2d.drawImage(img:pacman4right, pacmanx + 1, pacmany + 1, observer: this);
                break;
            default:
                // Si no coincide con ninguna de las opciones anteriores, dibujar la imagen predeterminada de Pacman
                g2d.drawImage(img:pacman1, pacmanx + 1, pacmany + 1, observer: this);
                break;
        }
    }
```

Paso 23

Este método se encarga de dibujar las paredes y los puntos del laberinto, creando la apariencia visual del escenario de juego.

```
//se realizo una sobreescritura de Metodos para dibujar drawMaze
//este método se encarga de dibujar las paredes y los puntos del laberinto,
//creando la apariencia visual del escenario de juego.
    private void drawMaze(Graphics2D g2d) {
        short i = 0;
        int x, y;
        // Recorrer el arreglo screendata para dibujar el laberinto
        for (y = 0; y < scrsize; y += blocksize) {
            for (x = 0; x < scrsize; x += blocksize) {
                g2d.setColor(c: mazecolor);
                g2d.setStroke(new BasicStroke(width: 2));
                // Verificar si hay una pared en la dirección izquierda (bit 1)
                if ((screendata[i] & 1) != 0) {
                    // Dibujar una línea vertical izquierda
                    g2d.drawLine(x1: x, y1: y, x2: x, y + blocksize - 1);
                }
                // Verificar si hay una pared en la dirección arriba (bit 2)
                if ((screendata[i] & 2) != 0) {
                    // Dibujar una línea horizontal superior
                    g2d.drawLine(x1: x, y1: y, x + blocksize - 1, y2: y);
                }
                // Verificar si hay una pared en la dirección derecha (bit 3)
                if ((screendata[i] & 4) != 0) {
                    g2d.drawLine(x + blocksize - 1, y1: y, x + blocksize - 1,
                            y + blocksize - 1); // Dibujar una línea vertical derecha
                }
                // Verificar si hay una pared en la dirección abajo (bit 4)
                if ((screendata[i] & 8) != 0) {
                    g2d.drawLine(x1: x, y + blocksize - 1, x + blocksize - 1,
                            y + blocksize - 1);// Dibujar una línea horizontal inferior
                }
                // Verificar si hay un punto en la celda (bit 5)
                if ((screendata[i] & 16) != 0) {
                    g2d.setColor(c: dotcolor);
                    // Dibujar un punto en la celda
                    g2d.fillRect(x + 11, y + 11, width: 2, height:2);
                }

                i++;
            }
        }
    }
```

Paso 24

El método initGame es responsable de configurar y establecer varios aspectos del estado del juego al inicio o después de reiniciar. Estos aspectos incluyen las vidas del jugador, la puntuación, el nivel, el número de fantasmas y la velocidad del juego.

```
/*el método initGame es responsable de configurar y restablecer varios aspectos del estado del juego
al inicio o después de reiniciar. Estos aspectos incluyen las vidas del jugador, la puntuación,
el nivel, el número de fantasmas y la velocidad del juego.
*/
   private void initGame() {

    pacsleft = 3;          // Inicializar la cantidad de vidas del jugador
    score = 0;             // Inicializar la puntuación del jugador
    initLevel();           // Inicializar el nivel de juego
    nrofghosts = 6;        // Establecer el número de fantasmas
    currentspeed = 3;      // Establecer la velocidad inicial del juego
    }
```

Paso 25

El método initLevel se encarga de preparar y configurar el nivel del juego, asegurándose de que los datos del laberinto sean correctos y luego llamando a continueLevel para seguir con la inicialización del nivel.

```
//El método initLevel se encarga de inicializar el nivel del juego, copiando los datos del laberinto desde
   //leveldata a screendata y luego continuando con la inicialización del nivel.
    private void initLevel() {
        // Copiar los datos del nivel actual desde leveldata a screendata
        int i;
        for (i = 0; i < nrofblocks * nrofblocks; i++) {
            screendata[i] = leveldata[i];
        }
        // Continuar con la inicialización del nivel
        continueLevel();
    }
```

Paso 26

El método continueLevel se encarga de configurar y establecer los atributos iniciales de los fantasmas y Pacman, así como de establecer varios aspectos del estado del juego cuando un nuevo nivel comienza.

```
//El método continueLevel se utiliza para continuar con la configuración y el
    //estado del juego después de inicializar un nuevo nivel.
    private void continueLevel() {
    short i;
    int dx = 1; // Dirección inicial de los fantasmas
    int random; // Variable para generar un número aleatorio
    // Configuración inicial de los fantasmas
    for (i = 0; i < nrofghosts; i++) {
        ghosty[i] = 4 * blocksize; // Posición vertical de los fantasmas
        ghostx[i] = 4 * blocksize; // Posición horizontal de los fantasmas
        ghostdy[i] = 0; // Dirección vertical de los fantasmas
        ghostdx[i] = dx; // Dirección horizontal de los fantasmas
        dx = -dx; // Invertir la dirección horizontal para el siguiente fantasma
        random = (int) (Math.random() * (currentspeed + 1)); // Generar un número aleatorio para la velocidad
        if (random > currentspeed) {
            random = currentspeed; // Limitar la velocidad aleatoria a la velocidad actual
        }
        ghostspeed[i] = validspeeds[random]; // Asignar la velocidad aleatoria a los fantasmas
    }
    pacmanx = 7 * blocksize; // Posición horizontal inicial de Pacman
    pacmany = 11 * blocksize; // Posición vertical inicial de Pacman
    pacmandx = 0; // Dirección horizontal de Pacman
    pacmandy = 0; // Dirección vertical de Pacman
    reqdx = 0; // Dirección horizontal deseada de Pacman
    reqdy = 0; // Dirección vertical deseada de Pacman
    viewdx = -1; // Dirección horizontal de la vista del jugador
    viewdy = 0; // Dirección vertical de la vista del jugador
    dying = false; // Bandera que indica si Pacman está muriendo
}
```

Paso 27

El método loadImages se encarga de cargar las imágenes necesarias para los elementos gráficos del juego, como los fantasmas y las animaciones de Pacman en diferentes direcciones y estados. Estas imágenes se cargan desde archivos ubicados en una carpeta llamada "imagenes".

```
//el método loadImages se encarga de cargar las imágenes necesarias para los elementos gráficos del juego,
//como los fantasmas y las animaciones de Pacman en diferentes direcciones y estados. Estas imágenes se
//cargan desde archivos ubicados en una carpeta llamada "imagenes".
    private void loadImages() {
        // Cargar la imagen del fantasma
        ghost = new ImageIcon(location: getClass().getResource(name: "../imagenes/ghost.gif")).getImage();
        // Cargar las imágenes de las animaciones de Pacman en diferentes direcciones y estados
        pacman1 = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman1.gif")).getImage();
        pacman2up = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman2up.gif")).getImage();
        pacman3up = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman3up.gif")).getImage();
        pacman4up = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman4up.gif")).getImage();
        pacman2down = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman2down.gif")).getImage();
        pacman3down = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman3down.gif")).getImage();
        pacman4down = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman4down.gif")).getImage();
        pacman2left = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman2left.gif")).getImage();
        pacman3left = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman3left.gif")).getImage();
        pacman4left = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman4left.gif")).getImage();
        pacman2right = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman2right.gif")).getImage();
        pacman3right = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman3right.gif")).getImage();
        pacman4right = new ImageIcon(location: getClass().getResource(name: "../imagenes/pacman4right.gif")).getImage();

    }
```

Paso 28

Utilizamos una sobreescritura paintComponent llamaremos el método de la clase base para así realizar la tarea de limpieza por medio del super. Llamaremos al método para realizar el dibujo de los componentes gráficos.

```
//sobre escritura
   @Override
public void paintComponent(Graphics g) {
   // Llamar al método paintComponent de la clase base para realizar tareas de limpieza
    super.paintComponent(g);
    // Llamar al método para realizar el dibujo de los componentes gráficos
    doDrawing(g);
}
```

Paso 29

Creamos el método principal para realizar el dibujo de los componentes gráficos utilizamos el objeto Graphics2D para acceder las funcionalidades avanzadas, utilizamos para el color, para dibujar el laberinto damos a conocer la puntuación del juego para realizar la animación, utilizamos una condición para para llamar la imagen del juego llamamos el método para dibujar el juego actual mostraremos la pantalla de la introducción, llamamos el dibujo de nuestro panel de la imagen.

```
// Método principal para realizar el dibujo de los componentes gráficos
//Estos métodos se sobrescriben en la clase tablero para proporcionar la lógica específica de dibujo.
private void doDrawing(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;// Convertir el objeto Graphics a Graphics2D para acceder a funcionalidades avanzadas
    // Establecer el color de fondo y llenar el panel
    g2d.setColor(c: Color.black);
    g2d.fillRect(x: 0, y: 0, width: d.width, height:d.height);
    // Dibujar el laberinto
    drawMaze(g2d);
    drawScore(g: g2d);// Dibujar la puntuación del juego
    doAnim();// Realizar animaciones
    if (ingame) {// Si el juego está en marcha.
        playGame(g2d);// Llamar a un método para dibujar el juego actual
    } else {
        showIntroScreen(g2d);// Mostrar la pantalla de introducción
    }
    g2d.drawImage(img:ii, x: 5, y: 5, observer: this);// Dibujar la imagen en el panel
    Toolkit.getDefaultToolkit().sync();// Sincronizar la imagen con el hardware gráfico
    g2d.dispose();// Liberar los recursos de gráficos
}
```