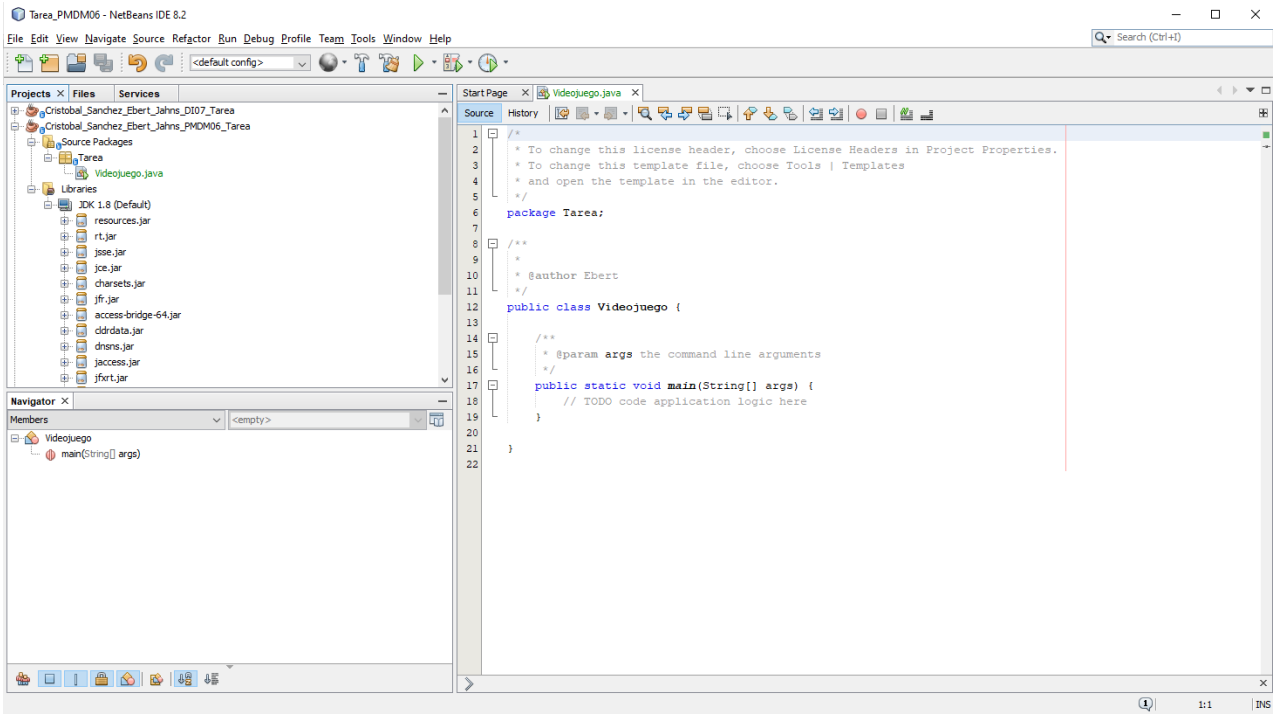
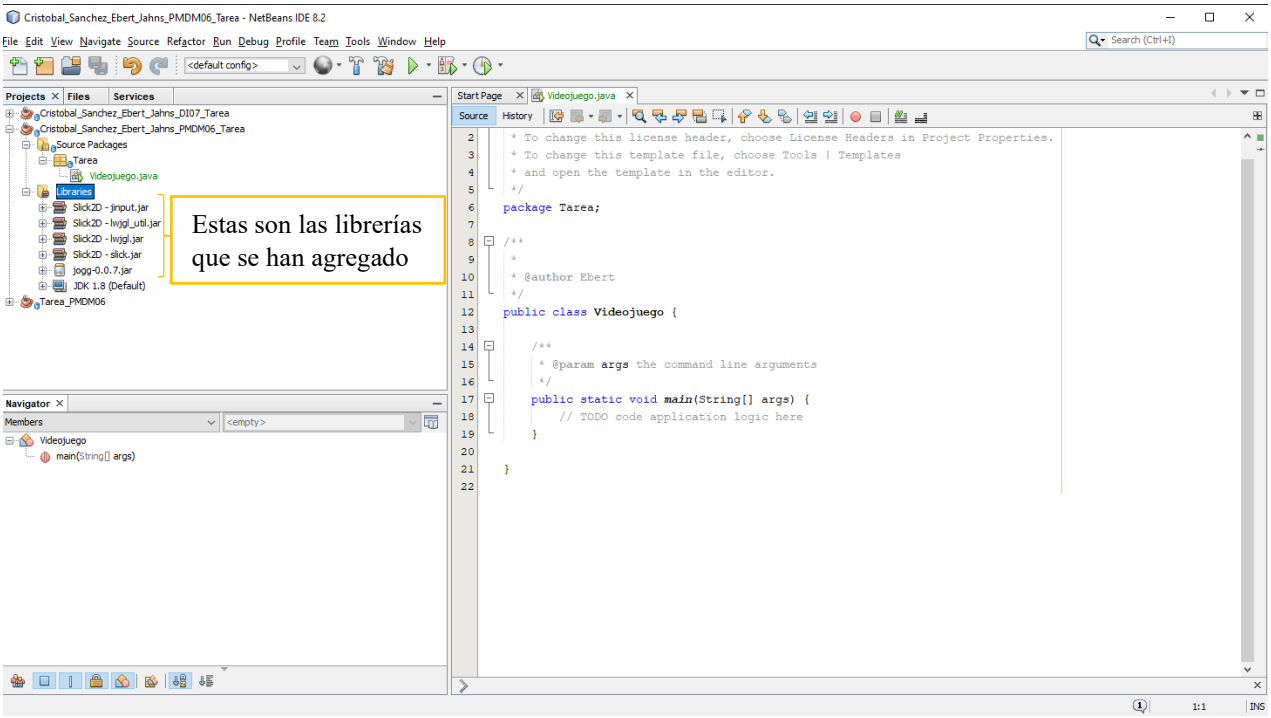
TAREA PARA PMDM06

Para realizar esta tarea se ha utilizado **NetBeans 8.2, JDK 1.8 y SO Windows 10**

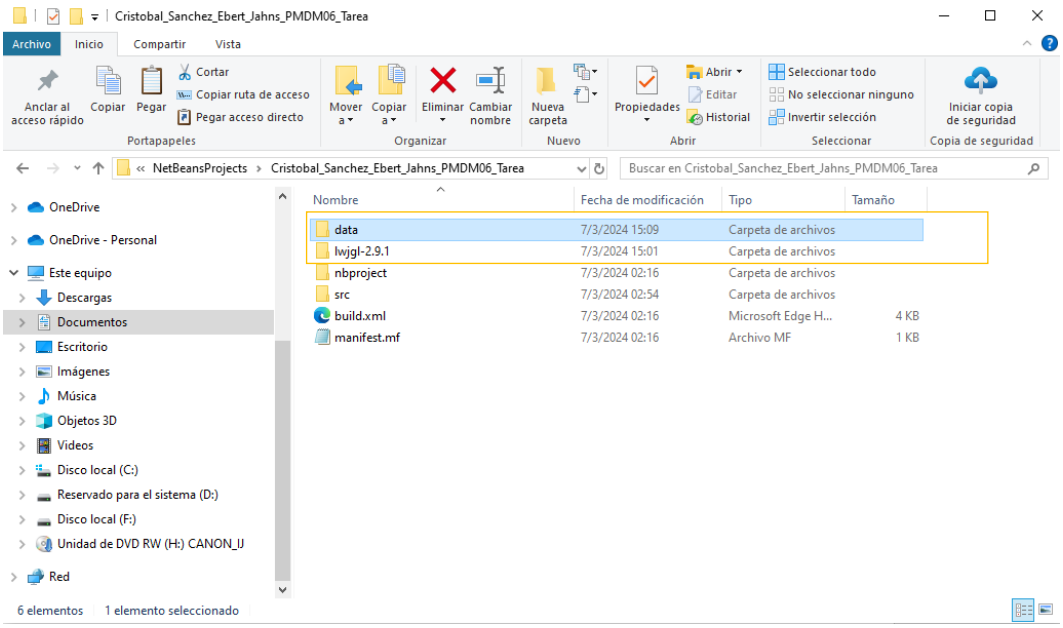
Primero se ha creado el proyecto **Cristobal_Sanchez_Ebert_Jahns_PMDM06_Tarea**.



Luego se ha descargado **Slick2D y lwjgl 2.9.1,** y se han agregado las librerías necesarias en el proyecto para [Configurar Slick2D en NetBeans – Notas de software], este enlace se encuentra en el temario de esta unidad.

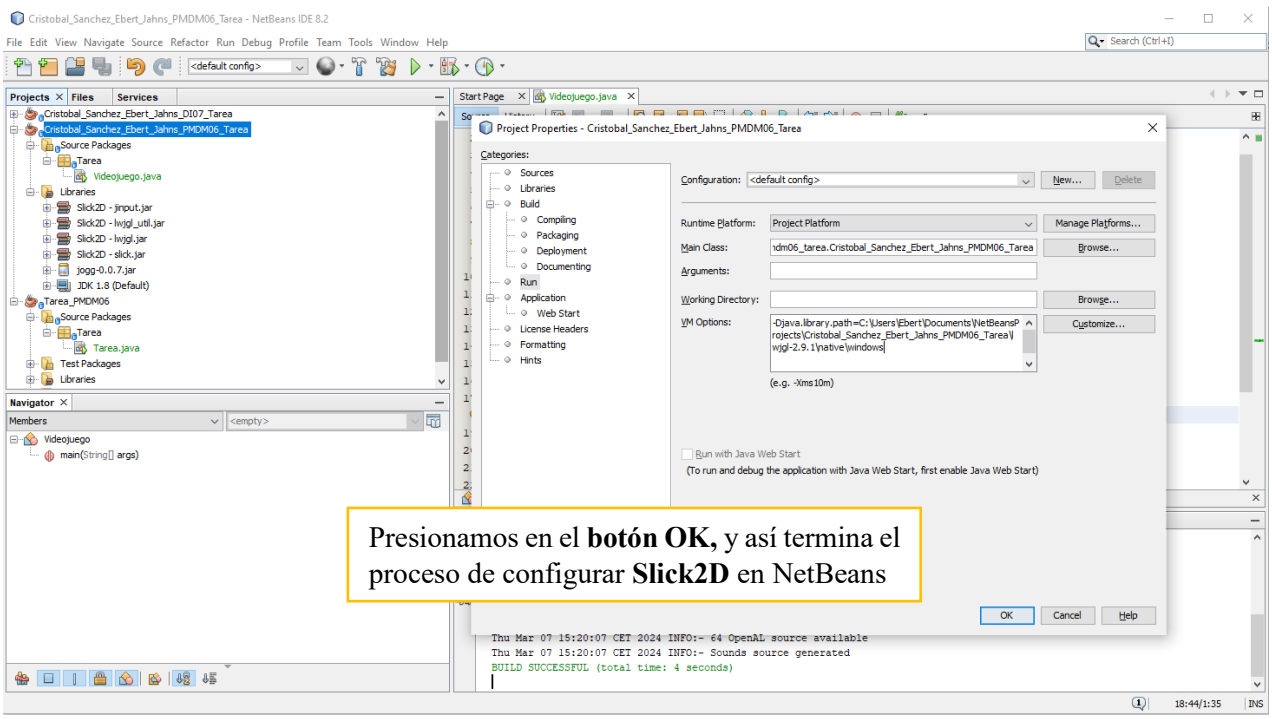


Dentro de la carpeta del proyecto he colocado el directorio **lwjgl 2.9.1** que se descargó anteriormente y el directorio **data,** donde **data** contiene sonidos, mapas, imágenes y recursos que se utilizaran en la tarea.

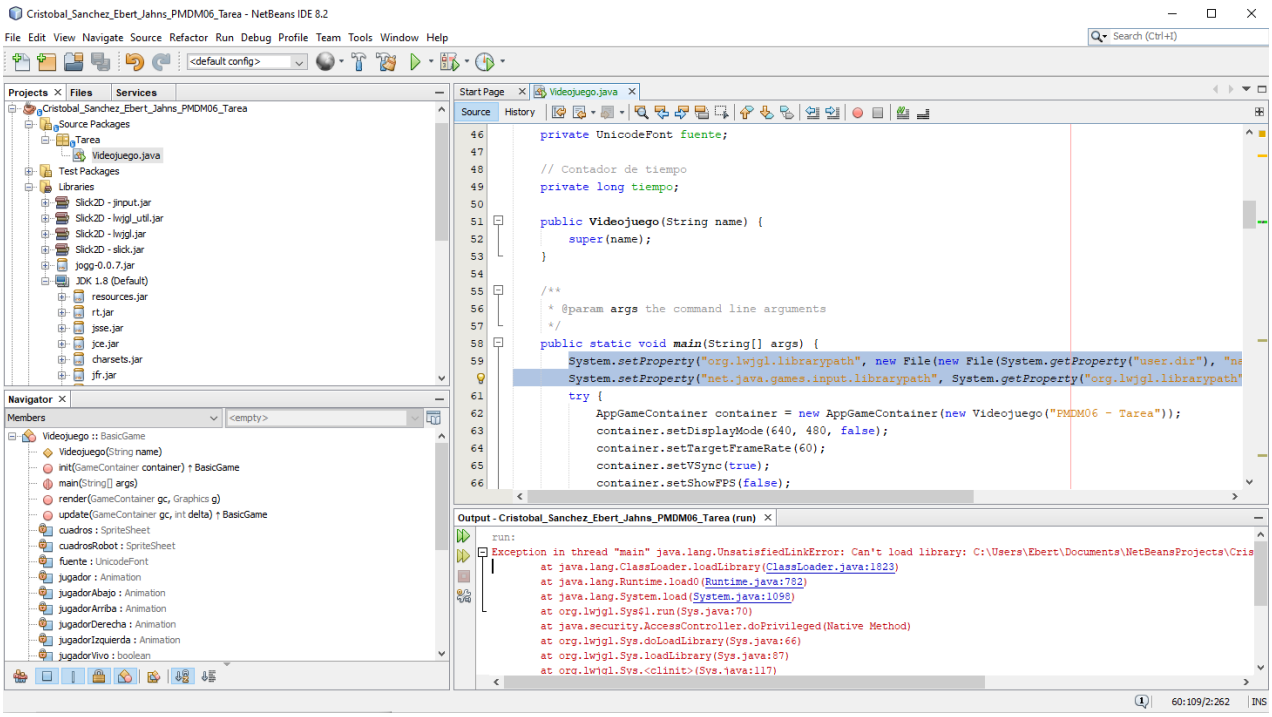


Es importante tener definida la ubicación del directorio **lwjgl 2.9.1** para realizar la configuración de los archivos nativos de **lwjgl.** Para ello se abre las propiedades del proyecto y seleccionamos **Run** y en **VM Options** colocamos:

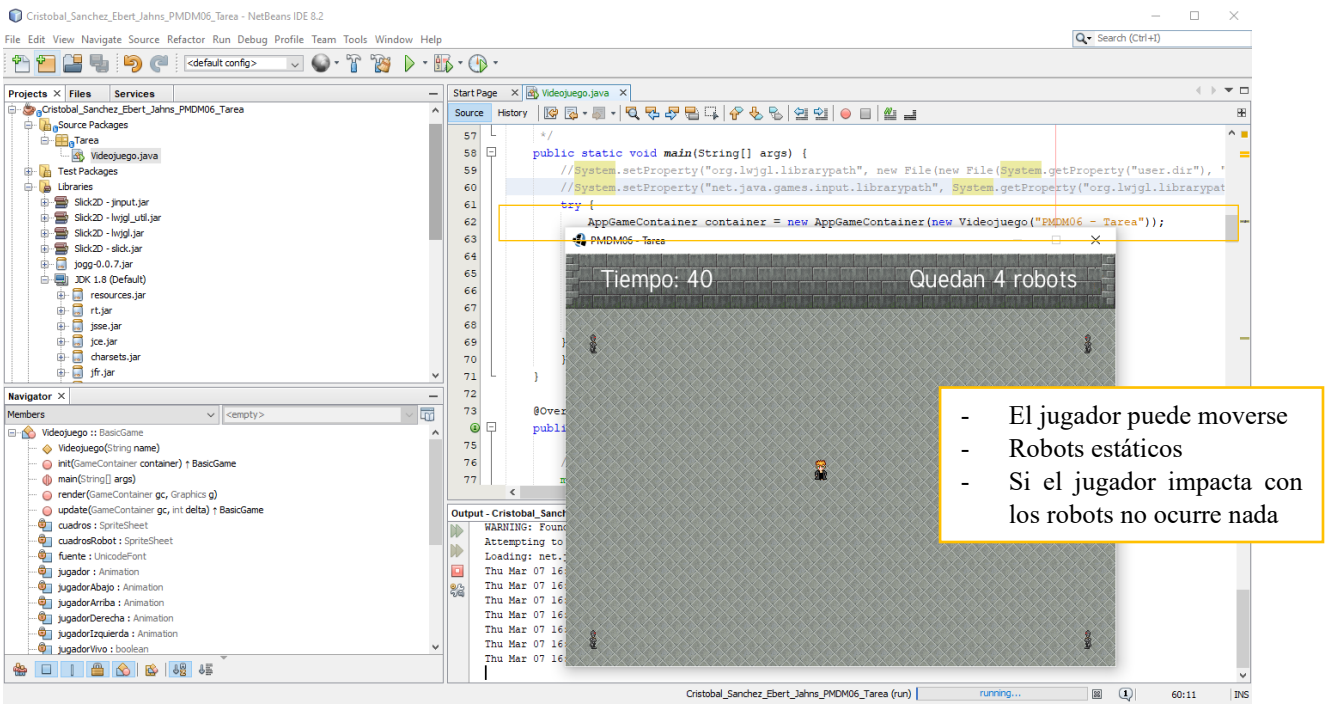
```
-Djava.library.path=C:\Users\Ebert\Documents\NetBeansProjects\Cristobal_Sanchez_Ebert_Jahns_PMDM06_Tarea\lwjgl-2.9.1\native\windows
```



Como punto de partida se ha utilizado el código proporcionado en los recursos de esta tarea, se copió el código en la clase **Videojuego.java**. Al ejecutar el código nos muestra el siguiente error:

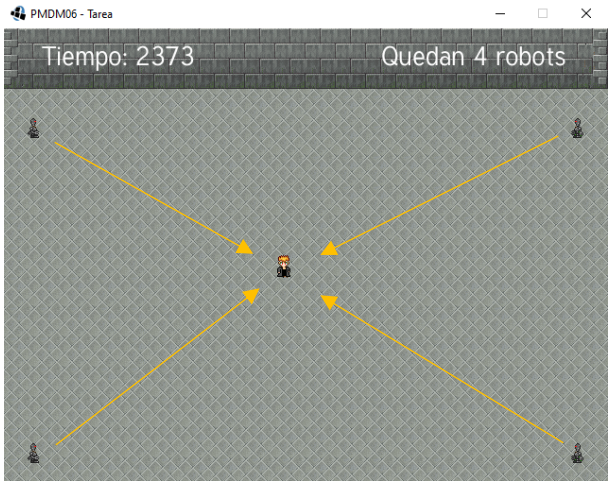


La única forma que encontré para poder solucionar este error fue comentar las dos líneas de código que se ven resaltadas en la imagen anterior. Al ejecutar el código se abre una ventana con el juego. Posteriormente se va a modificar el código para cumplir con los requerimientos que solicita esta tarea.

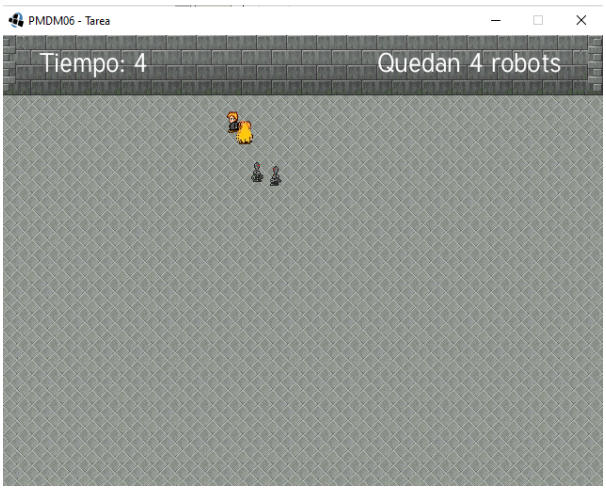


Reglas del juego:

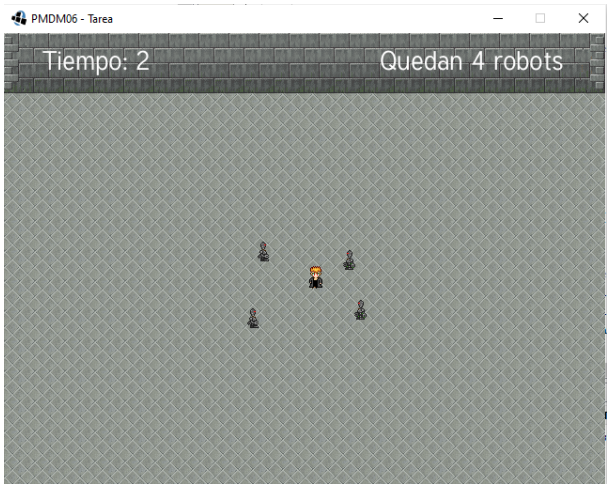
- **El jugador aparece en el centro y hay un grupo robots que lo rodean.**
  En este juego, un jugador se encuentra en el centro de un área de juego con robots ubicados en cada esquina. Cuando el juego comienza, los robots comienzan a moverse hacia el jugador siguiendo la dirección de las flechas. Sin embargo, si el jugador se mueve de su posición central, los robots cambiarán su dirección y comenzarán a perseguirlo. El objetivo es que el jugador evite ser atrapado por los robots mientras se desplaza estratégicamente por el área de juego.



- **Los robots explotan cuando colisionan con algo.**
- **El objetivo del jugador es eliminar a los robots haciéndolos chocar entre sí.**
  Estas dos reglas se cumplen cuando dos robots colisionan entre ellos, se encienden en llamas y están a punto de explotar. A pesar de estar en llamas, estos robots siguen persiguiendo al jugador. Sin embargo, cuando finalmente explotan, emiten un sonido de explosión y desaparecen del mapa. Esta adición añade un elemento de peligro adicional para el jugador.



- **Los robots se dirigen en línea recta hacia la posición del jugador.**
  Al inicio, los robots se mueven en línea recta hacia el jugador. Sin embargo, cuando el jugador se mueve, los robots cambian su objetivo y comienzan a perseguir al jugador. Entonces, la estrategia del jugador implica moverse de tal manera que pueda hacer que los robots colisionen entre sí, eliminándolos del juego.



- **El juego acabará cuando algún robot toque al jugador (pierde) o bien todos los robots sean eliminados (gana).**

  El jugador pierde
  Si un robot logra tocar al jugador, se muestra en pantalla **"Fin del juego"** y se reproduce un sonido de derrota. Esto indica que el jugador ha perdido y el juego ha terminado.

El jugador gana.
Si el jugador logra eliminar a todos los robots, se muestra en pantalla un mensaje que dice **"¡Has ganado!"** y se reproduce un sonido de victoria. Esto indica que el jugador ha ganado y el juego ha terminado.

Código completo del juego
A continuación, se va a mostrar el código completo. Al código se le han añadido los elementos necesarios para completar el juego. El código permite lo siguiente:

- Reproducir música de fondo, sonidos para cuando el jugador gana, pierde y cuando los robots explotan.
- Se agrego movimientos a los robots, para perseguir al jugador.
- Se agrego la animación para cuando colisionan los robots, pareciendo que se están incendiando
- Se modifico el código para que muestre el tiempo trascurrido, el tiempo se detiene cuando el juego termina y no se reinicia a 0.
- Se ha modificado el juego para que el jugador aparezca rodeado de 8 robots.

El código esta comentado, donde se explica para que sirve cada bloque de código.

- **Código Videojuego.java**

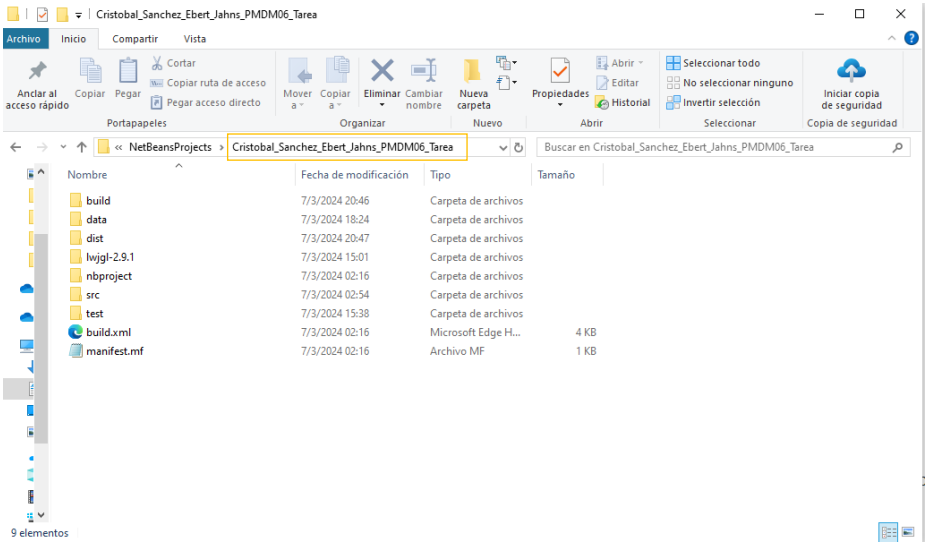
```
import org.newdawn.slick.*;
import org.newdawn.slick.font.effects.ColorEffect;
import org.newdawn.slick.geom.Rectangle;
import org.newdawn.slick.tiled.*;

    /**
     *
     * @author Ebert
     */
    public class Videojuego extends BasicGame {

        // Tilemap
        private TiledMap mapa;

        // Estado del jugador
        private float jugadorX, jugadorY;
        private SpriteSheet cuadros;
        private SpriteSheet cuadrosRobot;
        private SpriteSheet explosion;
        private Animation jugador;
        private Animation jugadorArriba;
        private Animation jugadorDerecha;
        private Animation jugadorAbajo;
        private Animation jugadorIzquierda;
        private Animation animacionColision;
        private boolean jugadorVivo;

        // Estado de los robots
        private Animation[] robot;
        private Animation robotArriba;
        private Animation robotDerecha;
        private Animation robotAbajo;
        private Animation robotIzquierda;
        private Animation robotArriba2;
        private Animation robotDerecha2;
        private Animation robotIzquierda2;
        private Animation robotAbajo2;
        private float[] robotX, robotY;
        private boolean[] robotVivo;
        private int numeroRobotsVivos;

        // Escritura de cadenas
        private UnicodeFont fuente;

        // Contador de tiempo
        private long tiempo;
        private long tiempoFinal;

        // Sonidos
        private Sound winSound;
        private Sound lostSound;
        private Sound boomSound;
        private boolean sonidoGanarReproduciendose = false;
        private boolean sonidoPerderReproduciendose = false;

        //Musica
        private Music musicaFondo;

        // Variables de estado del juego
        private boolean juegoGanado = false;
        private boolean juegoPerdido = false;

        public Videojuego(String name) {
            super(name);
        }

        /**
         * @param args the command line arguments
         */
        public static void main(String[] args) {
            //System.setProperty("org.lwjgl.librarypath", new File(new File(System.getProperty("user.dir"),
"native"), LWJGLUtil.getPlatformName()).getAbsolutePath());
            //System.setProperty("net.java.games.input.librarypath",
System.getProperty("org.lwjgl.librarypath"));
            try {
                AppGameContainer container = new AppGameContainer(new Videojuego("PMDM06 - Tarea"));
                container.setDisplayMode(640, 480, false);
                container.setTargetFrameRate(60);
                container.setVSync(true);
                container.setShowFPS(false);
                container.setUpdateOnlyWhenVisible(false);
                container.start();
            } catch (SlickException e) {
            }
        }

        @Override
        public void init(GameContainer container) throws SlickException {

            // Cargar el mapa
            mapa = new TiledMap("data/mapa.tmx", "data");
```

```
        // Inicializar la música de fondo
        musicaFondo = new Music("data/musica_de_fondo.wav");
        // Reproducir la música de fondo en bucle
        musicaFondo.loop();

        // Cargar los sonidos
        winSound = new Sound("data/ganaste.wav");
        lostSound = new Sound("data/perdiste.wav");
        boomSound = new Sound("data/disparo.wav");

        // Cargar las spritesheets
        cuadros = new SpriteSheet("data/heroe.png", 24, 32);
        cuadrosRobot = new SpriteSheet("data/robot.png", 24, 32);
        explosion = new SpriteSheet("data/heroe12.png", 24, 32);

        // Cargar animaciones del jugador
        jugadorArriba = new Animation(cuadros, 0, 0, 2, 0, true, 150, false);
        jugadorDerecha = new Animation(cuadros, 0, 1, 2, 1, true, 150, false);
        jugadorAbajo = new Animation(cuadros, 0, 2, 2, 2, true, 150, false);
        jugadorIzquierda = new Animation(cuadros, 0, 3, 2, 3, true, 150, false);
        jugador = jugadorAbajo;

        // Cargar animaciones del robot
        robotArriba = new Animation(cuadrosRobot, 0, 0, 2, 0, true, 150, true);
        robotDerecha = new Animation(cuadrosRobot, 0, 1, 2, 1, true, 150, true);
        robotAbajo = new Animation(cuadrosRobot, 0, 2, 2, 2, true, 150, true);
        robotIzquierda = new Animation(cuadrosRobot, 0, 3, 2, 3, true, 150, true);

        // Cargar  Nuevas animaciones
        robotArriba2 = new Animation(cuadrosRobot, 0, 0, 2, 0, true, 150, true);
        robotDerecha2 = new Animation(cuadrosRobot, 0, 1, 2, 1, true, 150, true);
        robotAbajo2 = new Animation(cuadrosRobot, 0, 2, 2, 2, true, 150, true);
        robotIzquierda2 = new Animation(cuadrosRobot, 0, 3, 2, 3, true, 150, true);

        // Cargar la animación de colisión
        animacionColision = new Animation(explosion, 0, 0, 2, 0, true, 150, true);

        // Estado inicial del jugador
        jugadorX = 320;
        jugadorY = 240;
        jugadorVivo = true;

        // Estado inicial de los robots
        robot = new Animation[8];
        robotX = new float[]{20, 596, 20, 596, 596, 20, 288, 288};
        robotY = new float[]{84, 84, 428, 428, 200, 200, 84, 428};
        robotVivo = new boolean[]{true, true, true, true, true, true, true, true};
        numeroRobotsVivos = 8;

        // Colocar los robots mirando al jugador en las 4 esquinas
        robot = new Animation[]{robotDerecha, robotIzquierda, robotDerecha,
                robotIzquierda, robotDerecha, robotIzquierda, robotDerecha, robotIzquierda};

        // Cargar el tipo de letra y los símbolos necesarios
        fuente = new UnicodeFont("data/tuffy.ttf", 28, false, false);
        fuente.addAsciiGlyphs();
        fuente.addGlyphs("áéíóúÁÉÍÓÚñÑ¡¿");
        fuente.getEffects().add(new ColorEffect(java.awt.Color.WHITE));
        fuente.loadGlyphs();

        // Comenzar la cuenta de tiempo
        tiempo = System.currentTimeMillis();
    }

    @Override
    public void update(GameContainer gc, int delta) throws SlickException {
        Input entrada = gc.getInput();

        // Si el jugador está muerto o no quedan robots, no actualizar nada
        if (!jugadorVivo || numeroRobotsVivos == 0) {
            return;
        }

        // Movimiento del jugador
        if (entrada.isKeyDown(Input.KEY_DOWN)) {   // Tecla abajo
            jugadorY += delta * 0.1f;
            // Limitar el movimiento dentro del contenedor
            if (jugadorY > (gc.getHeight() - jugador.getHeight())) {
                jugadorY = (gc.getHeight() - jugador.getHeight());
            }
            jugador = jugadorAbajo;
            jugador.update(delta);
        }
        if (entrada.isKeyDown(Input.KEY_UP)) { // Tecla arriba
            jugadorY -= delta * 0.1f;
            // Limitar el movimiento dentro del contenedor
            if (jugadorY < 32) {
                jugadorY = 32;
            }
            jugador = jugadorArriba;
            jugador.update(delta);
        }
        if (entrada.isKeyDown(Input.KEY_RIGHT)) {  // Tecla derecha
            jugadorX += delta * 0.1f;
```

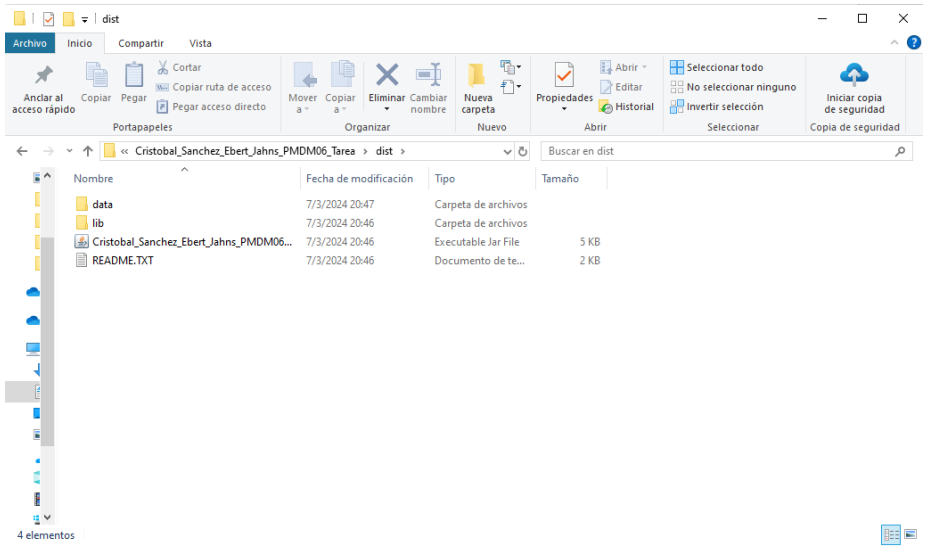
```
                // Limitar el movimiento dentro del contenedor
                if (jugadorX > (gc.getWidth() - jugador.getWidth())) {
                    jugadorX = (gc.getWidth() - jugador.getWidth());
                }
                jugador = jugadorDerecha;
                jugador.update(delta);
            }
            if (entrada.isKeyDown(Input.KEY_LEFT)) {    // Tecla izquierda
                jugadorX -= delta * 0.1f;
                // Limitar el movimiento dentro del contenedor
                if (jugadorX < 0) {
                    jugadorX = 0;
                }
                jugador = jugadorIzquierda;
                jugador.update(delta);
            }

            // Detectar colisión con los robots
            Rectangle jugadorRect = new Rectangle((int) (jugadorX + jugador.getWidth() * 0.1), (int)
(jugadorY + jugador.getHeight() * 0.1), (int) (jugador.getWidth() * 0.8), (int) (jugador.getHeight() * 0.8));
            for (int i = 0; i < 8; i++) {
                if (robotVivo[i]) {
                    // Calcular la distancia entre el jugador y el robot
                    float dx = jugadorX - robotX[i];
                    float dy = jugadorY - robotY[i];
                    float distancia = (float) Math.sqrt(dx * dx + dy * dy);

                    // Definir la distancia mínima para la colisión (10 píxeles)
                    float distanciaMinima = 10;

                    // Si la distancia entre el jugador y el robot es menor que la distancia mínima,
                    // consideramos que están lo suficientemente cerca como para estar en colisión
                    if (distancia < distanciaMinima) {
                        // Si hay colisión entre el jugador y un robot, el jugador muere
                        jugadorVivo = false;
                        lostSound.play();
                        sonidoPerderReproduciendose = true;
                        juegoPerdido = true;
                        tiempoFinal = System.currentTimeMillis();
                        return; // Salir del método update() ya que el jugador está muerto
                    }
                }
            }

            // Actualizar posición de los robots para que persigan al jugador
            for (int i = 0; i < 8; i++) {
                if (robotVivo[i]) {
                    // Calcular la dirección hacia la que debe moverse el robot para alcanzar al jugador
                    float dx = jugadorX - robotX[i];
                    float dy = jugadorY - robotY[i];
                    float distancia = (float) Math.sqrt(dx * dx + dy * dy);
                    // Normalizar el vector de dirección para que el robot se mueva a una velocidad constante
                    float velocidad = 0.1f;
                    dx /= distancia;
                    dy /= distancia;
                    // Mover el robot en la dirección calculada
                    robotX[i] += dx * velocidad * delta;
                    robotY[i] += dy * velocidad * delta;
                }

                // Verificar colisiones entre robots
                for (int j = i + 1; j < 8; j++) {
                    if (robotVivo[i] && robotVivo[j]) {
                        // Calcular la distancia entre los centros de los robots
                        float distanciaX = Math.abs(robotX[i] - robotX[j]);
                        float distanciaY = Math.abs(robotY[i] - robotY[j]);
                        double distanciaCentros = Math.sqrt(distanciaX * distanciaX + distanciaY *
distanciaY);

                        // Definir la distancia mínima para la colisión entre robots (ajustar según sea
necesario)
                        double distanciaMinima = 2; // Por ejemplo, 2 píxeles

                        // Si los robots están lo suficientemente cerca, colisión detectada
                        if (distanciaCentros < distanciaMinima) {
                            // Reproducir sonido de colisión
                            boomSound.play();
                            //Subir Volumen de la musica de fondo despues de la explosion
                            musicaFondo.setVolume(0.8f);

                            // Desactivar los robots que colisionaron
                            robotVivo[i] = false;
                            robotVivo[j] = false;

                            // Guardar el tiempo en que ocurrió la colisión
                            tiempoFinal = System.currentTimeMillis();

                            // Reducir el número de robots vivos
                            numeroRobotsVivos -= 2;

                        }
                    }
                }
            }
```

```
            // Comprobar colisiones adicionales entre robots
            for (int i = 0; i < 8; i++) {
                for (int j = i + 1; j < 8; j++) {
                    if (robotVivo[i] && robotVivo[j]) {
                        // Calcular la distancia entre los centros de los robots
                        float distanciaX = Math.abs(robotX[i] - robotX[j]);
                        float distanciaY = Math.abs(robotY[i] - robotY[j]);
                        double distanciaCentros = Math.sqrt(distanciaX * distanciaX + distanciaY *
distanciaY);

                        // Definir la distancia mínima para la colisión entre robots (ajustar según sea
necesario)
                        double distanciaMinima = 2; // Por ejemplo, 2 píxeles

                        // Si los robots están lo suficientemente cerca, colisión detectada
                        if (distanciaCentros < distanciaMinima) {
                            //Reducir volumen de la musica de fondo mientras el robot explota
                            musicaFondo.setVolume(0.05f);
                            // Cambiar la animación de los robots a la animación de colisión
                            robot[i] = animacionColision;
                            robot[j] = animacionColision;
                        }
                    }
                }
            }

            // Reproducir sonido de ganar si se han eliminado todos los robots
            if (numeroRobotsVivos == 0 && !sonidoGanarReproduciendose) {
                winSound.play();
                sonidoGanarReproduciendose = true;
                juegoGanado = true; // Indicar que el juego ha sido ganado
                tiempoFinal = System.currentTimeMillis();
            }

        }

        @Override
        public void render(GameContainer gc, Graphics g) throws SlickException {

            // Dibujar el mapa
            mapa.render(0, 0);

            // Dibujar al jugador
            jugador.draw(jugadorX, jugadorY);

            // Dibujar a los robots si están vivos
            for (int i = 0; i < 8; i++) {
                if (robotVivo[i]) {
                    robot[i].draw(robotX[i], robotY[i]);
                }
            }

            // Dibujar el número de robots restantes
            fuente.drawString(400, 10, "Quedan " + numeroRobotsVivos + " robots");

            // Mostrar mensajes de fin de juego si corresponde
            if (juegoGanado) {
                String mensajeGanador = "¡Has ganado!";
                fuente.drawString((gc.getWidth() - fuente.getWidth(mensajeGanador)) / 2, (gc.getHeight() -
fuente.getHeight(mensajeGanador)) / 2, mensajeGanador, Color.yellow);
            } else if (juegoPerdido) {
                String mensajePerdedor = "Fin de juego";
                fuente.drawString((gc.getWidth() - fuente.getWidth(mensajePerdedor)) / 2, (gc.getHeight() -
fuente.getHeight(mensajePerdedor)) / 2, mensajePerdedor, Color.red);
            }

            // Mostrar el tiempo transcurrido si el juego ha terminado
            if (juegoGanado || juegoPerdido) {
                // Detener la música de fondo cuando el juego se cierre
                musicaFondo.stop();
                // Calcular la duración del juego
                long duracionJuego = tiempoFinal - tiempo;
                // Dibujar el tiempo transcurrido
                fuente.drawString(40, 10, "Tiempo: " + duracionJuego / 1000);
            } else {
                // Mostrar el tiempo transcurrido si el juego sigue en progreso
                fuente.drawString(40, 10, "Tiempo: " + (System.currentTimeMillis() - tiempo) / 1000);
            }
        }

    }
```

Esta es la carpeta del proyecto



La carpeta dist contiene todos estos archivos, que son necesarios para ejecutar el archivo **.jar**



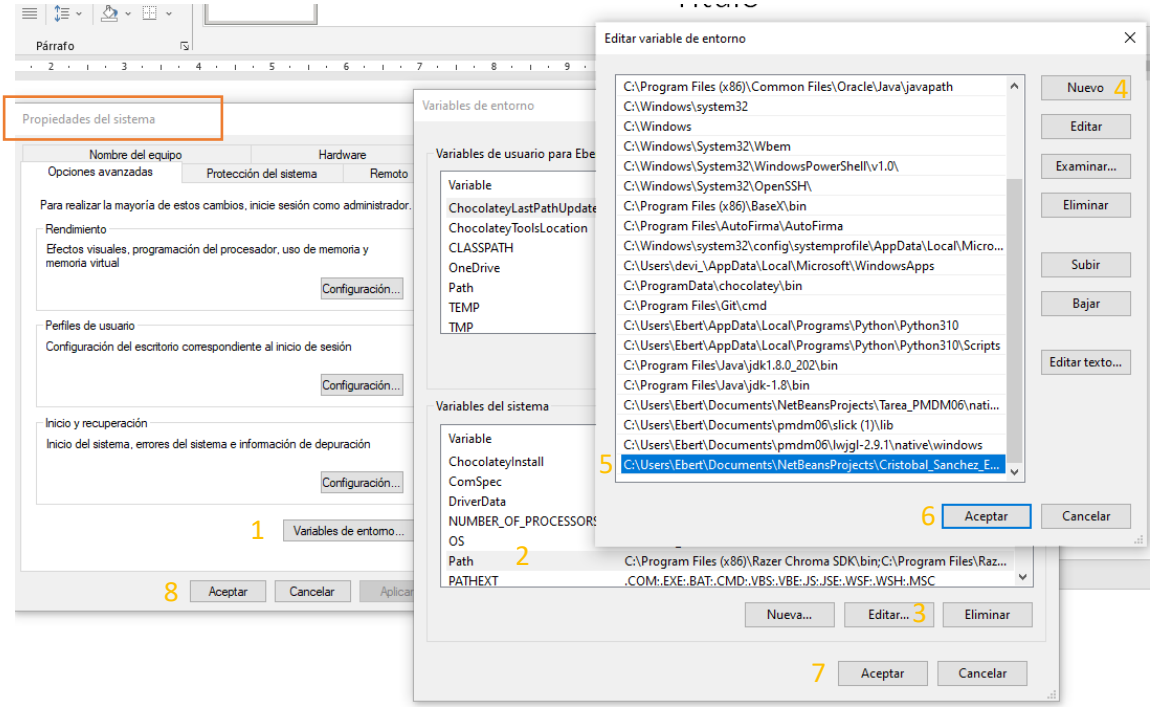
Si queremos probar el juego desde el archivo **.jar** y al ejecutar el archivo el juego no se inicia, es posible que el sistema operativo no pueda encontrar la ruta correcta de Java para ejecutar el programa. Para solucionar esto en Windows, debes configurar la variable de entorno **PATH** para que el sistema operativo pueda encontrar estas bibliotecas cuando intentes ejecutar el archivo **.jar**

Al agregar la ruta:

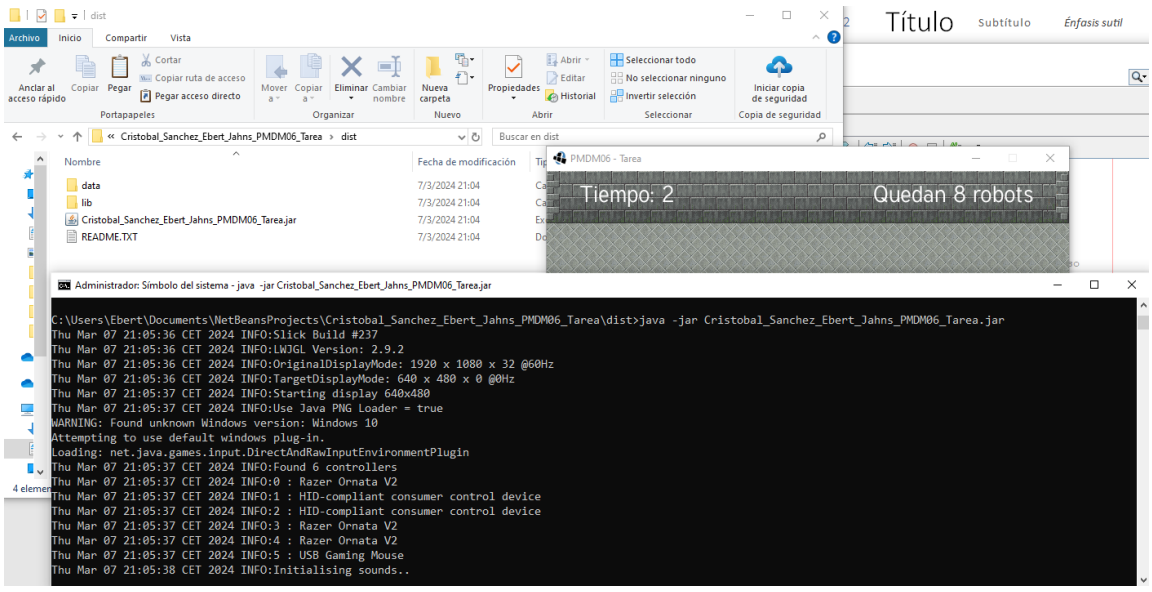
C:\Users\Ebert\Documents\NetBeansProjects\Cristobal_Sanchez_Ebert_Jahns_PMDM06_Tarea\lwjgl-2.9.1\native\windows

Donde: C:\Users\Ebert\Documents\NetBeansProjects es la ruta donde se ha colocado el proyecto.

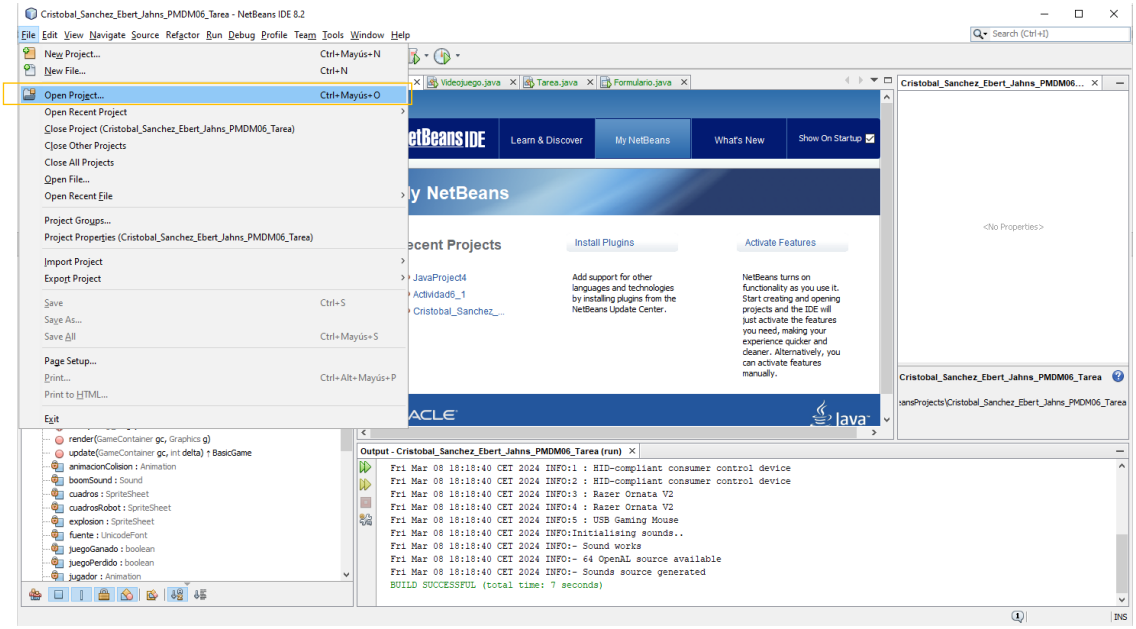
Se resuelve el problema al ejecutar el archivo **.jar**; sin embargo, ejecutar la aplicación directamente desde NetBeans no presenta ningún problema. Esto sugiere que el problema estaba relacionado con la configuración del **PATH** del sistema para encontrar las bibliotecas nativas de LWJGL. En la imagen siguiente se muestra donde se tiene que configurar el **PATH** con la ruta que hemos indicado anteriormente:



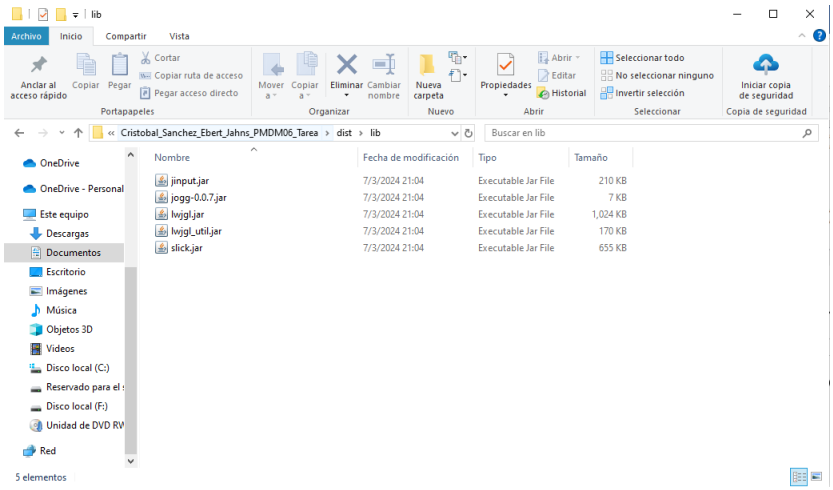
Ejecutar la aplicación desde la consola confirma que la configuración del **PATH** del sistema está funcionando correctamente y que la aplicación se ejecuta sin problemas fuera del entorno de desarrollo. **Importante cerrar la consola y volver abrirla para que pueda tomar los cambios realizados.**



Para finalizar, si se quiere probar el juego desde NetBeans se recomienda abrir el proyecto desde **Open Project…** y seleccionar la carpeta del proyecto **Cristobal_Sanchez_Ebert_Jahns_PMDM06_Tarea**



Al tener el proyecto ya montado en **NetBeans** seguro va a dar errores, pero es porque no se cargan las librerías, para ello hay que agregar las librerías que se encuentran en la ruta **Cristobal_Sanchez_Ebert_Jahns_PMDM06_Tarea\dist\lib**



Y con eso debería funcionar sin problemas.