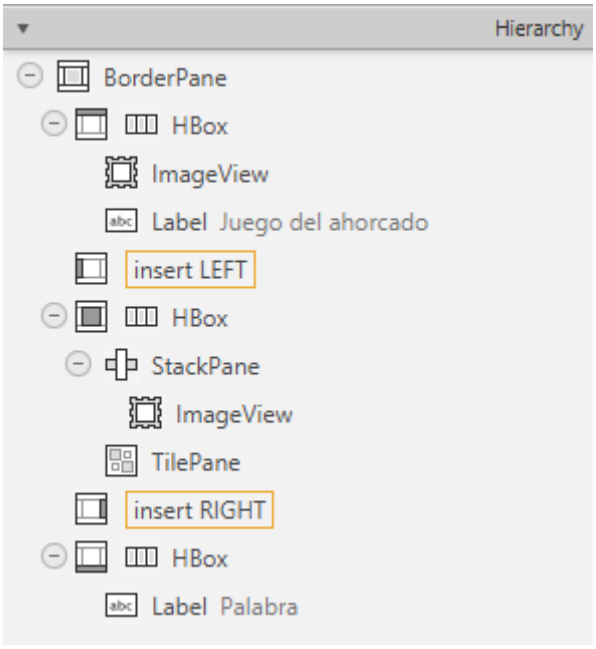
Índice

(pulsa en los titulos para ir a cada seccion)

1. Interfaz Visual

El primer paso es buscar el diseño que nos sea adecuado y acogedor. Yo he elegido como panel principal un ‘Border Pane’ con ‘Horizontal boxes’ en cada sector. Y en el sector central un panel para la imagen del ahorcado, y otro panel para almacenar los botones del juego. Esta sería la jerarquía de mis ‘containers’ y ‘controls’.



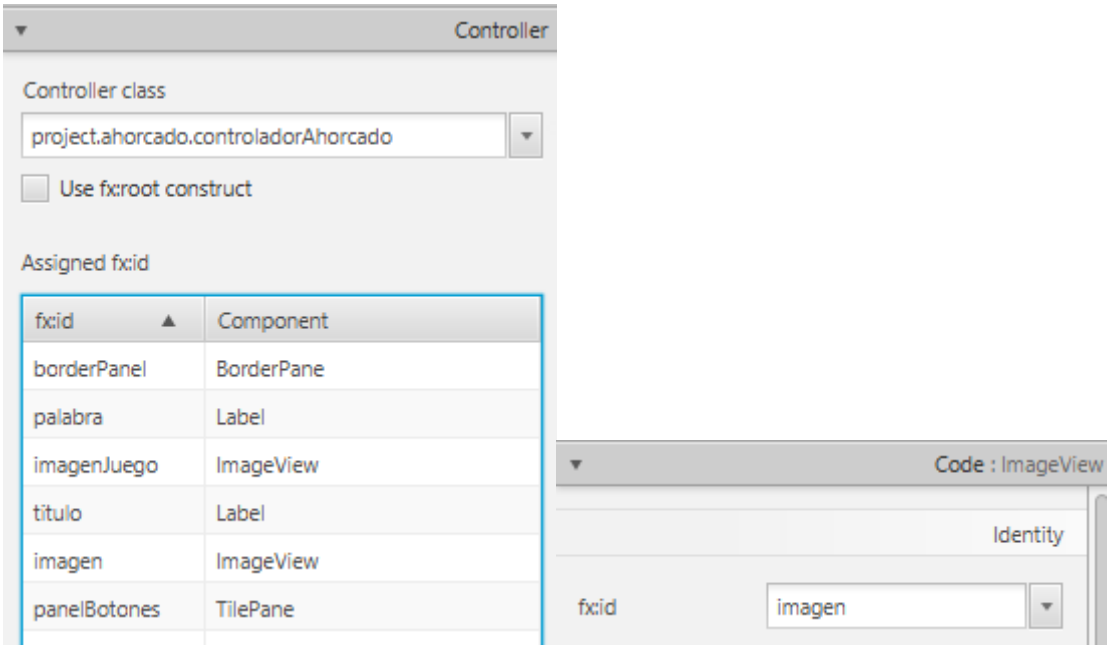
Y el diseño final usando estilos CSS seria este:

2. Enlazando los controles al código

Definimos los ‘controles’ y ‘containers’ en el controlador del juego.

```
@FXML
BorderPane borderPanel;

@FXML
TilePane panelBotones;

@FXML
Label titulo;

@FXML
Label palabra;

@FXML
ImageView imagen;

@FXML
ImageView imagenJuego;
```

Y enlazamos cada elemento en el SceneBuilder. Previamente habrá que haber enlazado el controlador del juego al fxml mediante SceneBuilder.



3. Colocación de los botones

Para la colocación de los botones no se debe usar SceneBuilder y poner todas las letras del alfabeto.

La idea sería utilizar un ‘bucle for’ recorriendo un string con todas las letras, ir creando los botones y almacenándolos en nuestro panel para los botones.

A la hora de crear el botón desactivo que pueda tener el ‘foco’ y le añado una acción cuando se pulsa que llama a un método que explicaré más tarde.

```
/**
 * Genera los botones de la A a la Z y los coloca en el panel.
 */
private void generarLetras() {
    String letras = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    panelBotones.setHgap(10);
    panelBotones.setVgap(10);
    panelBotones.setPrefColumns(5);
    panelBotones.setAlignment(Pos.CENTER);

    for (int i = 0; i < letras.length(); i++) {
        Button boton = new Button("" + letras.charAt(i));
        boton.setFocusTraversable(false);
        boton.setMinSize(30, 30);
        boton.setOnAction(acciones -> pulsarBoton(boton.getText(), boton));
        boton.getStyleClass().add("boton-letra");
        panelBotones.getChildren().add(boton);

    }
}
```

4. Funcionalidad del juego

Escoger la palabra secreta

El proyecto nos pide escoger una palabra al azar del documento proporcionado ‘palabras.txt’. Yo lo he implementado de esta manera:

```java
/**
 * Elige una palabra aleatoria del archivo palabras.txt.
 * Con su correción pertinente para poder abrirse cuando sea un .jar
 */
private void elegirPalabraRandom() {
    try {
        URI uri = controladorAhorcado.class
                .getResource("palabras.txt").toURI();
        Path path;

        if ("jar".equals(uri.getScheme())) {
            FileSystem fileSystem;
            try {
                fileSystem = FileSystems.getFileSystem(uri);
            } catch (FileSystemNotFoundException e) {
                fileSystem = FileSystems.newFileSystem(uri, Collections.emptyMap());
            }
            path = fileSystem.getPath("palabras.txt");
        } else {
            path = Paths.get(uri);
        }

        List<String> listaPalabras = Files.readAllLines(path);
        palabraAdivinar = listaPalabras
                .get(ThreadLocalRandom.current().nextInt(0, listaPalabras.size() + 1))
                .toUpperCase();

    } catch (URISyntaxException | IOException e) {
        System.out.println("Ruta no existe o error leyendo archivo: " + e.getMessage
());}
}
```

Este método se encarga de seleccionar una palabra aleatoria desde el archivo `palabras.txt`, ya sea que la aplicación se esté ejecutando desde un entorno de desarrollo o desde un archivo `.jar`. Para lograrlo, primero obtiene la ruta del archivo como recurso, y luego verifica si el programa se está ejecutando desde un `.jar`. Si es así, monta un sistema de archivos virtual para poder acceder al contenido comprimido del `.jar`, ya que no se puede leer directamente como un archivo normal. Finalmente, lee todas las palabras del archivo, elige una al azar y la convierte a mayúsculas. Esta lógica permite que el juego funcione correctamente en cualquier entorno de ejecución.

Esta variable palabraAdivinar la debemos de definir previamente como atributo de la clase:

```java
/** Palabra que el jugador debe adivinar. */
private String palabraAdivinar;
```

Ocultar las letras de la palabra secreta

Mediante el siguiente método recorremos la variable palabraAdivinar y la convertimos cada carácter a '_'. Termina asignando el nombre de la etiqueta donde se muestra la palabra codificada en el juego a la palabra codificada.

```java
/**
 * Codifica la palabra a adivinar en formato "_ _ _" para mostrar al jugador.
 */
private void codificarPalabra() {
    String palabraAdivinarCodificada = "";
    for (int i = 0; i < palabraAdivinar.length(); i++) {
        palabraAdivinarCodificada += "_ ";
    }
    palabra.setText(palabraAdivinarCodificada);
}
```

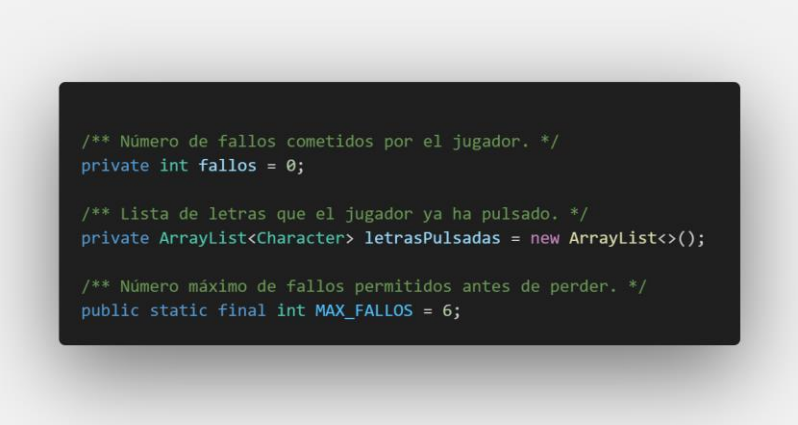
Lógica del juego

Para la lógica del juego he implementado el método pulsarBoton:

Este método se ejecuta cada vez que el jugador pulsa una letra del abecedario en el juego del ahorcado. Primero, desactiva el botón de la letra pulsada para que no se pueda volver a usar, y añade esa letra a una lista de letras ya jugadas. Luego, verifica si la letra está contenida en la palabra que se debe adivinar. Si la letra está presente, se reconstruye la palabra parcialmente revelada mostrando las letras acertadas y guiones bajos para las letras aún no descubiertas. Si no está, incrementa el contador de fallos y actualiza la imagen del ahorcado.

También se comprueba si el usuario ha ganado o ha perdido y ejecuta su respectivo código que explicaré ahora.

Tendremos la necesidad de crear estos atributos de clase para este método:



```java
/** Número de fallos cometidos por el jugador. */
private int fallos = 0;

/** Lista de letras que el jugador ya ha pulsado. */
private ArrayList<Character> letrasPulsadas = new ArrayList<>();

/** Número máximo de fallos permitidos antes de perder. */
public static final int MAX_FALLOS = 6;
```

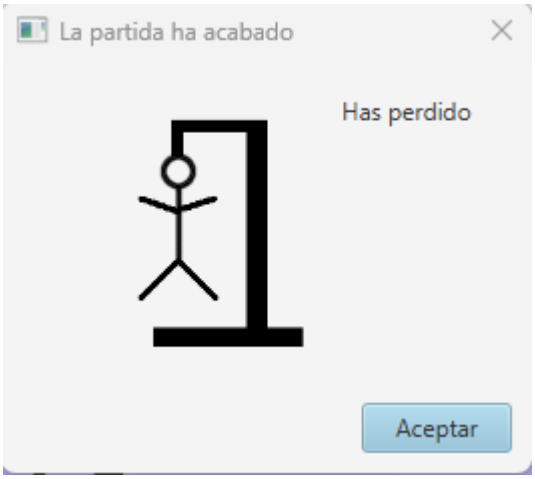
Código del método pulsarBoton:

```java
/**
 *
 * Método que se ejecuta al pulsar un botón/letra.
 *
 * @param c     Letra pulsada.
 * @param boton Botón correspondiente (se desactiva tras usarse).
 */
private void pulsarBoton(String c, Button boton) {
    boton.getStyleClass().remove("boton-letra");
    boton.getStyleClass().add("boton-pulsado");
    boton.setDisable(true);
    boolean acertada = true;
    letrasPulsadas.add(c.charAt(0));

    if (palabraAdivinar.contains(c)) {
        String formaSecreto = "";
        for (int i = 0; i < palabraAdivinar.length(); i++) {
            if (letrasPulsadas.contains(palabraAdivinar.charAt(i))) {
                formaSecreto += palabraAdivinar.charAt(i) + " ";
            } else {
                acertada = false;
                formaSecreto += "_ ";
            }
        }
        palabra.setText(formaSecreto.trim());
    } else {
        fallos++;
        acertada = false;
        imagen.setImage(new Image(controladorAhorcado.class
                .getResourceAsStream("Hangman-" + fallos + ".png")));
    }

    if (acertada) {
        alertaInformacion("La partida ha acabado", null, "Has ganado", "has_ganado.png");
        alertaVolverAJugar();
    }

    if (fallos == MAX_FALLOS) {
        alertaInformacion("La partida ha acabado", null, "Has perdido, la palabra era " +
 palabraAdivinar,
                "Hangman-6.png");
        alertaVolverAJugar();
    }
}
```

5. Cuadros de dialogo

Cuadro de dialogo informativo

Estos cuadros de dialogo en este caso son aquellos que salen cuando el usuario ha ganado o perdido una partida y son únicamente informativos. Como por ejemplo en el proyecto he hecho estos dos:



Esto en JavaFX se les llama Alertas, estas alertas pueden contener texto, imágenes etc...

Para esta alerta he hecho un método en común para ambas y así no repetir código:

```java
/**
 * Muestra una alerta informativa con imagen personalizada.
 *
 * @param title      Título de la alerta.
 * @param header     Texto del encabezado (puede ser null).
 * @param context    Texto del contenido principal.
 * @param imagenPath Ruta de la imagen a mostrar.
 */
private void alertaInformacion(String title, String header, String context, String imagenPath) {
    Alert alerta = new Alert(AlertType.INFORMATION);
    alerta.getDialogPane().setPrefSize(100, 80);
    alerta.setTitle(title);
    alerta.setHeaderText(header);
    alerta.setContentText(context);

    Image imagen = new Image(controladorAhorcado.class.getResourceAsStream(imagenPath));
    ImageView vista = new ImageView(imagen);
    vista.setFitWidth(150);
    vista.setFitHeight(150);
    alerta.setGraphic(vista);

    alerta.showAndWait();
}
```

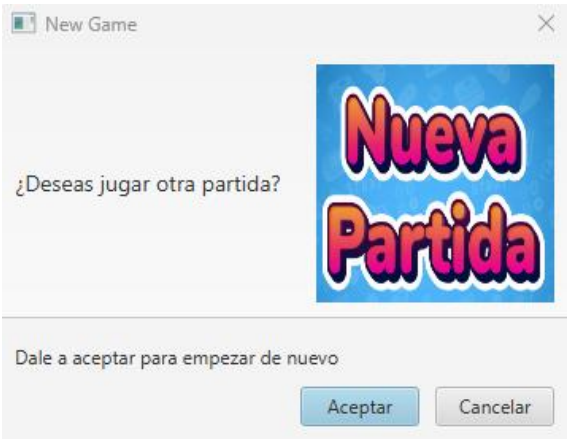
A este método se le pasan como parámetros, de cada sección el texto que se quiere mostrar y la ruta de la imagen a mostrar. He implementado este método de la siguiente manera:

```
    if (acertada) {
        alertaInformacion("La partida ha acabado", null, "Has ganado",
"has_ganado.png");
        alertaVolverAJugar();
    }

    if (fallos == MAX_FALLOS) {
        alertaInformacion("La partida ha acabado", null,
"Has perdido, la palabra era " + palabraAdivinar,
                "Hangman-6.png");
        alertaVolverAJugar();
    }
}
```

Al acertar la palabra saldrá la ventana de victoria y al llegar a los fallos máximos aparecerá la ventana de derrota.

Cuadro de dialogo confirmativo

Estos cuadros de dialogo en este caso son aquellos que salen después de que el usuario haya perdido o ganado y le pregunta si quiere hacer otra partida nueva o no. Además, también sale cuando queremos salir del juego mediante el botón de exit de la aplicación. Estos cuadros de dialogo son de elegir opción.



Para la alerta de volver a jugar he hecho un método:

```java
/**
 * Muestra una alerta de confirmación preguntando si el jugador quier
e jugar de
 * nuevo.
 * Si acepta, reinicia el juego. Si no, cierra la aplicación.
 */
private void alertaVolverAJugar() {
    Alert alert = new Alert(AlertType.CONFIRMATION);
    alert.setTitle("New Game");
    alert.setHeaderText("¿Deseas jugar otra partida?");
    alert.setContentText("Dale a aceptar para empezar de nuevo");

    Image imagen = new Image(controladorAhorcado.class.
getResourceAsStream("partida_nueva.png"));
    ImageView vista = new ImageView(imagen);
    vista.setFitWidth(150);
    vista.setFitHeight(150);
    alert.setGraphic(vista);

    Optional<ButtonType> resultado = alert.showAndWait();
    if (resultado.isPresent() && resultado.get() == ButtonType.OK) {
        reiniciarJuego();
    } else {
        // Menos forzoso que un System.exit(0);
        ((Stage) borderPanel.getScene().getWindow()).close();
    }
}
```

Es igual que la informativa pero esta alerta espera a que el usuario le dé a cancelar o a aceptar y en base a lo que pulse hará una cosa u otra. En este caso, reiniciar la partida o no.

Para reiniciarla he diseñado el siguiente método:

```java
/**
     * Reinicia el juego completamente, eligiendo nueva palabra y limpiand
o estado.
     */
    private void reiniciarJuego() {
        fallos = 0;
        letrasPulsadas.clear();
        panelBotones.getChildren().clear();
        initialize();
    }
```

Este método reestablece las variables necesarias, reestablece la imagen del hombre ahorcado, escoge otra palabra y vuelve a generar los botones.

Para la alerta de salir de la aplicación debes de capturar el evento de cerrar la ventana de la siguiente manera:

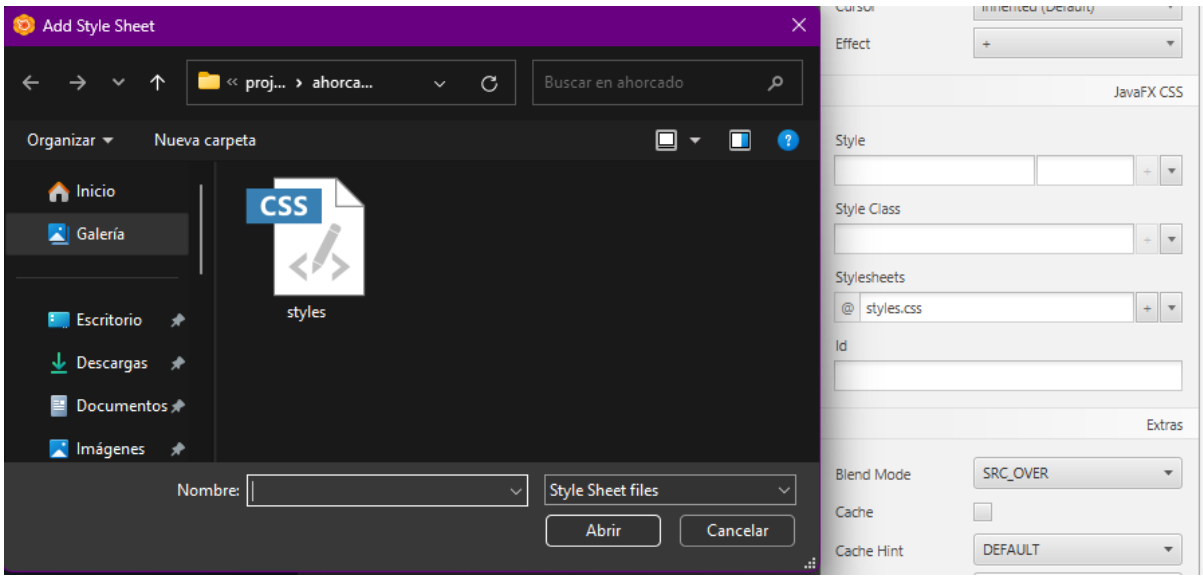
```java
@Override
public void start(Stage stage) throws IOException {
    scene = new Scene(loadFXML("ahorcado"), 500, 850);
    stage.setScene(scene);
    stage.setTitle("El ahorcado - Juan");
    stage.setMinHeight(800);
    stage.setMinWidth(425);
    stage.setOnCloseRequest(ventana -> {
        Alert alertaCierre = new Alert(AlertType.CONFIRMATION);
        alertaCierre.setTitle("Exit");
        alertaCierre.setHeaderText("Intentando salir. Se perderá el progreso");
        alertaCierre.setContentText("Quieres salir del juego?");
        Image imagen = new Image(controladorAhorcado.class.getResourceAsStream("adios.png"));
        ImageView vista = new ImageView(imagen);
        vista.setFitWidth(150);
        vista.setFitHeight(150);
        alertaCierre.setGraphic(vista);
        Optional<ButtonType> resultado = alertaCierre.showAndWait();
        if (resultado.isPresent() && resultado.get() != ButtonType.OK) {
            ventana.consume();
        }
    });
    stage.show();
```

Con stage.setOnCloseRequest capturamos el evento de cierre de ventana y usamos una expresión lambda para que ejecute ese bloque de código en ese momento. El resto es igual a la anterior, pero cambiando que si el usuario presiona aceptar la ventana se consume y se cierra la aplicación.

6. Hoja de estilos CSS

Como enlazar el CSS a nuestro FXML

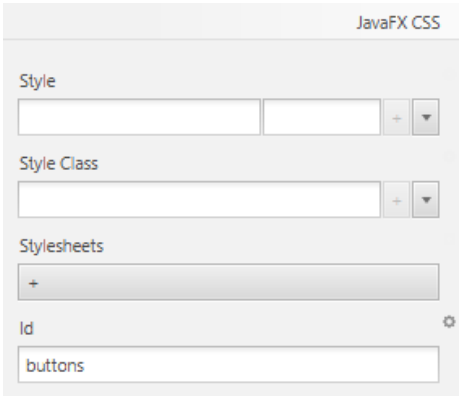
En el apartado de properties del container que engloba a todos los elementos agregamos la hoja de estilos de esta manera:



Id’s

Existen dos maneras de agregar estilo a ciertas partes de los containers:

Mediante id’s



En este apartado le agregamos un id al elemento al que queramos darle estilo y en el css ya lo editamos:

```
#buttons {
    -fx-background-color: #00ADB5;
    -fx-border-color: black;
    -fx-border-width: 5;
    -fx-border-radius: 20;
    -fx-background-radius: 27;
    -fx-padding: 5;
    -fx-background-insets: 0;
    -fx-border-insets: 0;


}
```

Este método lo he usado para:

```css
#bottom {

    -fx-border-color: black #222831 #222831 #222831;
    -fx-border-width: 5px;
    -fx-background-color: #222831;

}

#top {
    -fx-border-color: #222831 #222831 black #222831;
    -fx-border-width: 5px;
    -fx-background-color: #222831;
}

#buttons {
    -fx-background-color: #00ADB5;
    -fx-border-color: black;
    -fx-border-width: 5;
    -fx-border-radius: 20;
    -fx-background-radius: 27;
    -fx-padding: 5;
    -fx-background-insets: 0;
    -fx-border-insets: 0;


}

#word {
    -fx-text-fill: white;
    -fx-font-family:'Gill Sans', 'Gill Sans MT'
, Calibri, 'Trebuchet MS', sans-serif;
    -fx-font-size: 30px;
    -fx-font-weight: bold;


}

#title {
    -fx-text-fill: white;
    -fx-font-family:'Gill Sans', 'Gill Sans MT'
, Calibri, 'Trebuchet MS', sans-serif;
    -fx-font-size: 20px;
    -fx-font-weight: bold;

}
```

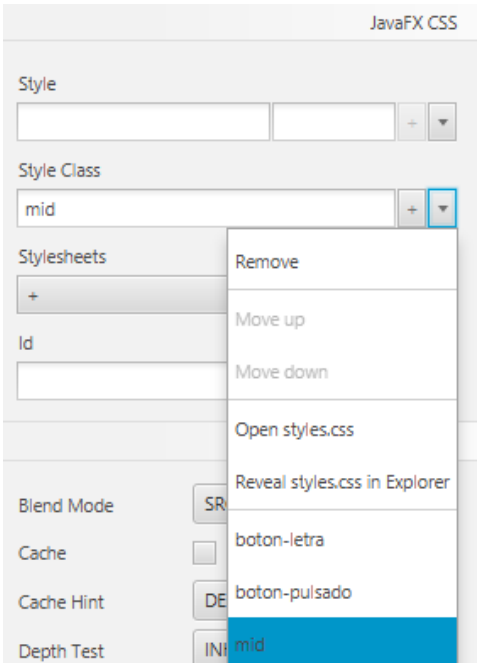
Clases

También se puede hacer mediante clases, para este método debemos seguir los siguientes pasos:

Primero hay que declarar la clase en el css de esta manera .nombreDeLaClase { codigo css }

```css
.mid {
    -fx-background-color: #393E46;
}
```

Y después, en SceneBuilder seleccionamos el elemento al que queremos añadir esa clase de css de la siguiente manera:



Y ya tendríamos el estilo enlazado. Este método lo he utilizado también para los botones, pero esta vez un poco diferente. En vez de hacerlo con SceneBuilder, lo enlazas mediante código ya que no puedes seleccionarlos en la escena:

```
boton.getStyleClass().add("boton-letra");
```

También para cuando el botón ha sido pulsado sea invisible:

```
private void pulsarBoton(String c, Button boton) {
        boton.getStyleClass().remove("boton-letra"
);      boton.getStyleClass().add("boton-pulsado");
```

Y el código css para cada uno sería el siguiente:

```
.boton-letra {

    -fx-background-color: black;
    -fx-text-fill: white;
    -fx-font-size: 16px;
    -fx-border-color: black;
    -fx-border-width: 2;
    -fx-border-radius: 5;
    -fx-background-radius: 5;
    -fx-padding: 5;
    -fx-background-insets: 0;
    -fx-border-insets: 0;

}

.boton-pulsado {
    -fx-background-color: #00ADB5;
    -fx-text-fill: #00ADB5;
    -fx-font-size: 16px;
    -fx-border-color: #00ADB5;
    -fx-border-width: 2;
    -fx-border-radius: 5;
    -fx-background-radius: 5;
    -fx-padding: 5;
    -fx-background-insets: 0;
    -fx-border-insets: 0;
}
```

De este modo los botones se verán de la siguiente manera: