FINAL PROJECT OOP

Game developer
Nguyen Huu Hoang Nam
Cao Ngoc Anh Tuan
Le Pham Thuy An
Dao Huu Hoai

Advisor
Ph.D. Le Duy Tan

Table of Contents

TETRIS

CHAP 1: INTRODUCTION

1.1 Overview of Tetris

Tetris, first developed by Alexey Pajitnov in 1984, is one of the most iconic puzzle games in history. The game involves fitting falling tetrominoes into a grid to complete horizontal lines, which are then cleared. Tetris has been adapted into numerous formats, captivating players for decades due to its simplicity and addictiveness.

1.2 Purpose of the Project

The purpose of this project is to recreate the classic Tetris experience while incorporating additional features such as the 'hold' mechanism and a preview of upcoming tetrominoes ('next' feature). This project aims to deepen understanding of game development using Java, enhance problem-solving skills, and apply advanced programming concepts such as object-oriented programming (OOP), graphical user interfaces (GUIs), and algorithm design.

1.3 Developer team

Name	Student's ID	Contribute
Le Pham Thuy An	ITCSIU23049	- Write report - Design UI - Create Starting interface - Testing
Dao Huu Hoai	ITCSIU23054	- Leader - Init Repository - Create Setting interface - Handle music setting - Manage github workflow - Fix bug - Testing - Task management - Draw UML
Nguyen Huu Hoang Nam	ITCSIU23028	- Handle Game Logic - Create Choose_Level interface - Create Playing interface - Testing - Handle animation - Analyzing project - Write report
Cao Ngoc Anh Tuan	ITCSIU21244	- Write report - Create Win game interface - Create Game over interface - Testing

CHAP 2: SOFTWARE REQUIREMENTS

2.1 What we have:

1. Friendly, efficient UI/UX design

2. Useful members and task manager system.

3. Easy to operate.

4. Measured coding and professional thinking.

2.2 What we want:

1. Develop interesting small game.

2. Use professional infrastructure to manage whole system (MVC)

3. Make it easy to maintain and update.

4. Enhance reading new technical documents skill

5. Enhance coding skills

6. Enhance skill and experience in finding and debugging system

2.3 Working tools, platform:

1. Intellij Idea / Vscode.

2. Figma design.

3. Github.

4. Trello.

5. draw.io

2.4 Use Case Scenario

We have created the use cases based on the UX view of the game.

Tetris game	Play game	Choose level	Playing game at chosen level
			Back to home
			Open setting
			Pause Game
			Reload Game
			Get win or lose game screen
	Setting	Setting the volumne of the theme music	

2.5 Use case diagram

Forward	Backward

2.6 Class diagram

TetrisLaucher

+ widthScreen: boolean

+ heightScreen: boolean

- musicController: MusicController

+ start(stage: Stage): void

+ setMusicVolume (volume: double): void

+ main (args: String[]): void

MusicController

- mediaPlayer: MediaPlayer

- volume: double

+ MusicController(volume: double): void

+ playMusic(): void

+ setMusicVolume(volume: double): void

<<Interface>>
BackwardScreen

+ backToPrevScreen(): void
+ backToMainScreen(): void

ViewController

- previousScene: Scene

- stage: Stage

+ ViewController (stage: Stage): void

+ getMainView(): void

+ getSettingView(): void

+ getChooseLevelView(): void

+ getPlayingView(level: String): void

+ getWinGameView(): void

+ getLoseGameView(): void

+ getStage(): Stage

+ setStage(stage: Stage): void

+ getPreviousScene(): Scene

+ setPreviousScene(previousScene:

TetrisGame

- width: int

- height: int

- grid: Color[][]

- currentTetrimino: Tetrimino

- gc: GraphicContext

- score: int

- scorePlus: int

+ TetrisGame(width: int, height: int, gc: GraphicContext)

+ createRandomTetrimino(): Tetrimino

+ getCurrentTetrimino(): Tetrimino

+ setCurrentTetrimino(tetrimino

+ canMoveTo(blocks: int[][]): boolean

+ moveTetrimino(dx: int, dy: int):

+ rotateTetrimino(): void

+ lockTetrimino(): void

+ clearFullRows(): void

+ clearRow(row: int): void

+ getScore(): int

+ render(): void

+ getScorePlus(): int

+ setScorePlus(): int

Tetrimino

- rotations: int[][][]

- rotationState: int

- color: Color

- posX: int

- posY: int

+ COLORS: Color[]

+ SHAPES: int[][][]

+ Tetrimino(type: int, x: int, y: int)

+ getBlocks(): int[][]

+ getRelativeBlocks(): int[][][]

+ getColor(): Color

+ move(dx: int, dy: int): void

+ rotate(): void

+ rotateBack(): void

SettingViewController

- musicSlider: Slider

- backBtnIcon: ImageView

- musicIcon: ImageView

+ onMusicSliderChange(): void

+ view(stage: Stage): void

+ initialize(): void

<<implements>> BackWardScreen

MainViewController

- startingImage: ImageView

+ view(stage: Stage): void

+ initialize(): void

+ onNewGameButtonClick(): void

+ onSettingButtonClick(): void

+ getStartingImage(): ImageView

+ setStartingImage(startingImage: ImageView): void

ChooseLevelViewController

- backArrowIcon: ImageView

+ view(stage: Stage): void

+ initialize(): void

+ clickBackBtnHanlder(): void

+ clickEasyBtnHanlder(): void

+ clickNormalBtnHanlder(): ImageView

+ clickHardBtnHanlder(): void

LoseGameViewController
- prevLevel: String
+ view(stage: Stage): void
+ getPrevLevel(): String
+ setPrevLevel(prevLevel: String): void
+ handleTryAgainButtonClick(): void

WinGameViewController
- musicSlider: Slider
- backBtnIcon: ImageView
- musicIcon: ImageView
+ handleClickNextLevelButton(): void
+ view(stage: Stage): void

PlayingViewController

- gameCanvas: Canvas
- homeIcon: ImageView
- settingIcon: ImageView
- clear30Line: ImageView
- reloadIcon: ImageView
- actionIcon: ImageView
- pauseIcon: Image
- continueIcon: Image
- scoreLabel: Label
- currentCanvas: Canvas
- nextCanvas1: Canvas
- nextCanvas2: Canvas
- nextCanvas3: Canvas
- tetrisGame: TetrisGame
- AnimationTimer: gameLoop
- nextTetrimino: List<Tetrimino>
- isPause: boolean

+ view(stage: Stage): void

+ initialize(): void

+ lockAndSpawnTetrimino(): void

+ renderNextTetriminos(): void

+ renderCurrentPlay(): void

+ drawTetrimino(tetrimino: Tetrimino, gc: GraphicContext, offsetX: double, offsetY: double): void

+ handleToggleActionIcon(): void

+ handleKeyPress(event: KeyEvent):

+ handleReloadGame(): void

+ openSettingView(): void

+ getGameLoop(): AnimationTimer

<<implements>> BackWardScreen

MainMenu

+ displayView(stage: Stage): void

LoseGame

+ score: SimpleIntegerProperty

+ createLoseScreen(stage: Stage): void
- createGreenBar(): Hbox
- createColumn(height: int): Vbox
- createSquare(): Rectangle
+ display(stage: Stage): void

ChooseLevelMenu

+ displayView(stage: Stage): void

WinGame

+ score: SimpleIntegerProperty

+ createWinScreen(stage: Stage): void
- createGreenBar(): Hbox
- createColumn(height: int): Vbox
- createSquare(): Rectangle
+ display(stage: Stage): void
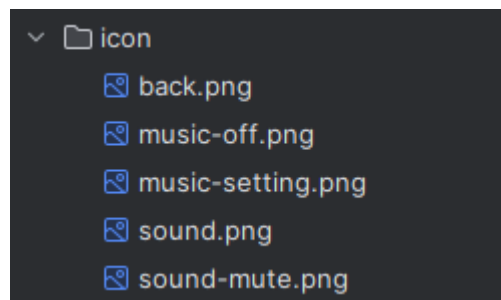
CHAPTER 3:
DESIGN & IMPLEMENTATION

3.1 UI Design

Resources files will be stored in main folder. We can put them in different directory with diffrent categories:
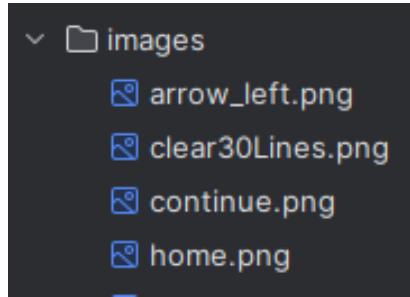
- **oopprj.tetrisgame.tetris:** contain fxml file which is used to define the user interface in JavaFX, separating the UI design from the application logic.
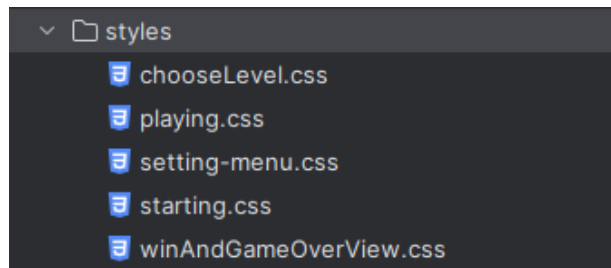

- **Icon:** contain any type of images that represent a symbol

- **Images:** background, button images,.... — most of the images for UI

- **Styles:** css files which is used for styling

With UI design, fxml file is main feature. So using FXML, we can quickly design UI layouts and the screen elements they contain, in the same way creating web pages in HTML — with a series of nested elements.

Each layout file must have exactly one root element. After defining the root, we can add other layout objects or widgets as child elements to construct a View hierarchy for our layout. For instance, an FXML layout using a VBox—a layout container that organizes its child nodes in a vertical column while automatically positioning them based on their order and available space.

```xml
<?xml version="1.0" encoding="UTF-8"?>



<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<?import javafx.scene.image.ImageView?>
<?import javafx.scene.text.Font?>
<AnchorPane xmlns="http://javafx.com/javafx"
            xmlns:fx="http://javafx.com/fxml"
            fx:controller="controllers.SettingViewController"
            stylesheets="@../../../assets/styles/setting-menu.css">
    <Button styleClass="back-button" onAction="#backToPrevScreen">
        <graphic>
            <ImageView fx:id="backBtnIcon" fitWidth="40" fitHeight="40"/>
        </graphic>
    </Button>

        <Label styleClass="tittle" text="Settings" layoutY="50">
            <font>
                <Font name="Arial" size="40"/>
            </font>
        </Label>
        <ImageView fx:id="musicIcon" fitWidth="50" fitHeight="50" layoutX="50" layoutY="200"/>
        <Slider fx:id="musicSlider" min="0" max="100" value="50" blockIncrement="1" layoutX="120" layoutY="230" onMouseDragged="#onMusicS

</AnchorPane>
```

We can call css resource in fxml language by this method:

```
stylesheets="@../../../assets/styles/setting-menu.css">
```

CSS in FXML is used to style and customize the appearance of UI components, providing a way to separate the visual design from the layout and logic.

By setting **styleClass**, For example:

```xml
<Button text="EASY"
        styleClass="choose_level_button"
        onAction="#clickEasyBtnHandler"
/>
```

We can link it to the class with the same name in css file:

```css
.choose_level_button {
    -fx-pref-width: 200px;
    -fx-pref-height: 50px;
    -fx-background-color: #0D2E43;
    -fx-text-fill: white;
    -fx-font-size: 16px;
    -fx-background-radius: 16px;
    -fx-cursor: hand;
}
```
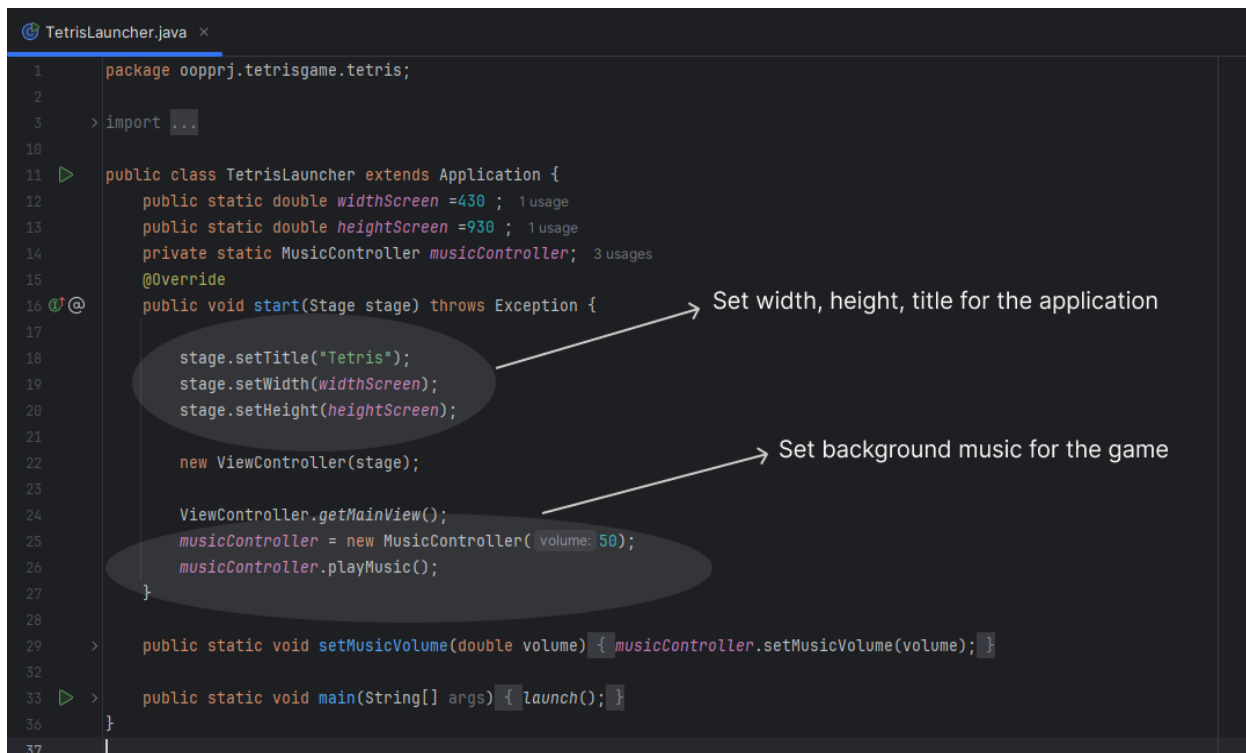
From this class, we can set some styles such as width, height, background color for the button

3.2 Main feature

3.2.1 Tetris Game Application

a.   Launcher

- **Application Launch:** TetrisLauncher is the entry point of the application. It extends Application and overrides the start method to set up the main stage with width, height and start background music.

```java
package oopprj.tetrisgame.tetris;

import ...

public class TetrisLauncher extends Application {
    public static double widthScreen =430 ;  1 usage
    public static double heightScreen =930 ;  1 usage
    private static MusicController musicController;  3 usages
    @Override
    public void start(Stage stage) throws Exception {

        stage.setTitle("Tetris");                      // Set width, height, title for the application
        stage.setWidth(widthScreen);
        stage.setHeight(heightScreen);

        new ViewController(stage);

        ViewController.getMainView();                  // Set background music for the game
        musicController = new MusicController( volume: 50);
        musicController.playMusic();
    }

    public static void setMusicVolume(double volume) { musicController.setMusicVolume(volume); }

    public static void main(String[] args) { launch(); }
}
```

- **Main Interface Setup:** ViewController is initialized with the main Stage. It has methods like getMainView, getSettingView, getChooseLevelView, and getPlayingView to switch between different scenes.

```java
ViewController.java  ×
1      package controllers;
2
3    > import ...
7
8      public class ViewController {
9          public static Scene previousScene;   3 usages
10
11         private static Stage stage;   14 usages
12
13   >     public ViewController(Stage stage) { ViewController.stage = stage; }
16
17   >     public static void getMainView() throws Exception {...}
20
21   >     public static void getSettingView() throws Exception {...}
26
27   >     public static void getChooseLevelView() throws Exception {...}
32
33 @ >     public static void getPlayingView(String level) throws Exception {...}
51
52   >     public static void getWinGameView() throws Exception {...}
55
56   >     public static void getLoseGameView() throws Exception {...}
59
```

Each method will implement the method displayView() which lies in each file in folder views through controller, and the related file will get the resource of fxml file and set the corresponding Scene on the Stage.

```java
public static void view(Stage stage) throws Exception {      PlayingViewController have a method to
    PlayingView.displayView(stage);                          implement displayView()
}
```

```java
package views;

> import ...

public class PlayingView {  2 usages
    public static void displayView(Stage stage) throws Exception {  1 usage
        try {
            FXMLLoader fxmlLoader = new FXMLLoader(TetrisLauncher.class.getResource( name: "playing-view.fxml"));
            Scene scene = new Scene(fxmlLoader.load());
            stage.setScene(scene);
            stage.show();
            scene.getRoot().requestFocus();
            PlayingViewController controller = fxmlLoader.getController();
            scene.setOnKeyPressed( KeyEvent e -> controller.handleKeyPress(e));
        }
        catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Load fxml file

Each fxml file have corresponding controller file to handle the event and and manages logic.

```xml
<VBox xmlns="http://javafx.com/javafx/11.0.1"
      xmlns:fx="http://javafx.com/fxml"
      fx:controller="controllers.ChooseLevelViewController"
      alignment="CENTER"
      stylesheets="@../../../assets/styles/chooseLevel.css">
```

b.   UI components

Our application using Javafx which offers pre-built components such as Button, Label, ImageView for building complex interfaces.

c.   Initialize Image

When we have a image on fxml, we need to map and initialize them. For example, our project has a image like this:

15

```xml
<ImageView fx:id="startingImage"
           fitHeight="100.0"
           fitWidth="100.0"
           preserveRatio="true">
```

To map this one with a variable in controller class we just need to cast its id, then initialize with the resource by giving the path to the image, so the fxml can render them.

```java
@FXML
private ImageView startingImage;
```

```java
@FXML
public void initialize() {
    startingImage.setImage(new Image(getClass().getResource( name: "/assets/images/startingImage.png").toExternalForm()));
}
```

d.   Sound & Animation Effect

- Animation

Animation in our JavaFX Tetris project is primarily managed using the AnimationTimer class, which creates a game loop to update and render the game state at regular intervals. The game loop handles the movement of Tetrimino pieces, rendering the game grid, and responding to user input through key events. This creates a smooth and interactive gaming experience.

By checking the condition:

(now - lastUpdate >= 500_000_000) (500_000_000 is counted in nano second) we can make our Tetris Game re-render after 0.5 second, it means that the tetrimino will

```java
gameLoop = new AnimationTimer() {  ± Nam +1
    private long lastUpdate = 0;  2 usages

    @Override  ± Nam +1
    public void handle(long now) {
        if (now - lastUpdate >= 500_000_000) {
            if (!tetrisGame.moveTetrimino( dx: 0,  dy: 1)) {
                try {
                    lockAndSpawnTetrimino();
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
            tetrisGame.render();
            lastUpdate = now;
        }
    }
};
```

be moved down one block after that interval.

- Sound

Similar with animation effect, in this case, we have MediaPlayer (This class is

the primary API for playing sound and video)

```
LoseGame.java        ChooseLevelMenu.java        SettingViewController.java        PlayingViewController.java        PlayingView.java

1    package controllers;
2
3    import javafx.scene.media.Media;
4    import javafx.scene.media.MediaPlayer;
5
6    public class MusicController {   ± "hwHoai"
7        private MediaPlayer mediaPlayer;   5 usages
8        private double volume;   3 usages
9
10   >    public MusicController(double volume) { this.volume = volume; }
13
14       public void playMusic() {   1 usage   ± "hwHoai"
15           Media media = new Media(getClass().getResource( name: "/au/theme.mp3").toExternalForm());
16           mediaPlayer = new MediaPlayer(media);
17           mediaPlayer.setVolume(volume);
18           mediaPlayer.setCycleCount(MediaPlayer.INDEFINITE);
19           mediaPlayer.play();
20       }
21
22       public void setVolume(double volume) {   1 usage   ± "hwHoai"
23           this.volume = volume;
24           mediaPlayer.setVolume(volume);
25       }
26   }
```

e.   Event Handler

- Key Event

KeyEvent in JavaFX represents keyboard interactions and occurs during key press),
release or typing events. It allows handling user input for actions like navigation,
shortcuts, or text input in an application.

In our case, we use it to control the movement and rotation of Tetrimino pieces during
gameplay

```
@FXML  1 usage   ± Nam +1
public void handleKeyPress(KeyEvent event) {
    System.out.println("Key pressed: " + event.getCode());
    if (isPaused) return;

    switch (event.getCode()) {
        case LEFT -> tetrisGame.moveTetrimino( dx: -1,  dy: 0);
        case RIGHT -> tetrisGame.moveTetrimino( dx: 1,  dy: 0);
        case DOWN -> tetrisGame.moveTetrimino( dx: 0,  dy: 1);
        case UP -> tetrisGame.rotateTetrimino();
    }
    tetrisGame.render();
}
```

-Mouse Event

A **MouseEvent** in JavaFX represents mouse interactions, such as clicks, movements, presses, releases, drags, and scrolls, allowing to handle user input involving the mouse

By using this and combine with Slider, we can control the volume of the music in our scenario

```java
//If the user change the music volume, the music icon will change
public void onMusicSliderChange(MouseEvent mouseEvent) {  1 usage  ≛ hwHoai +1
    if(musicSlider.getValue() == 0) {
        musicIcon.setImage(new Image(getClass().getResource( name: "/assets/icon/music-off.png").toExternalForm()));
    } else {
        musicIcon.setImage(new Image(getClass().getResource( name: "/assets/icon/music-setting.png").toExternalForm()));
    }

    System.out.printf("Music volume: %.2f\n", musicSlider.getValue());

    TetrisLauncher.setMusicVolume(musicSlider.getValue());
}
```

- Button Event

The FXML file defines the buttons and links them to event handler methodsin the Controller class using the onAction or onMouseClicked attributes to perform various actions such as navigating between screens, reloading the game, and toggling the game state (pause/resume).

```xml
<Region HBox.hgrow="ALWAYS" />
<Button onAction="#openSettingView" styleClass="btn">
```

```java
public void openSettingView() throws Exception {  1 usage
    ViewController.setPreviousScene(ViewController.getStage().getScene());
    ViewController.getSettingView();
}
```

f. Rendering

- Canvas

In our JavaFX Tetris project, the Canvas components are used to render the game grid, current Tetrimino, and next Tetrimino pieces.

The PlayingViewController class manages these canvases stage.

```
@FXML
private Canvas gameCanvas;
@FXML
private Canvas currentPlayCanvas;
@FXML
private Canvas nextCanvas1;
@FXML
private Canvas nextCanvas2;
@FXML
private Canvas nextCanvas3;
```

- Color

Colors are used to visually distinguish between different types of Tetrimino pieces. Each Tetrimino type is assigned a specific color, which

is defined in the COLORS array in the Tetrimino class.

The TetrisGame class uses these colors to render the game grid and the Tetrimino pieces on the canvas, enhancing the visual appeal and making it easier for players to identify the pieces..

```
public static final Color[] COLORS = {  1 usage
        Color.web( s: "#00FFFF"), Color.web( s: "#488BD4"),
        Color.web( s: "#00FF00"), Color.web( s: "#FF7F00"),
        Color.web( s: "#FFFF00"), Color.web( s: "#C200C2"),
        Color.web( s: "#D41E3C")
};
```

3.2.2 Main feature

a. Handle music

In our Tetris game, music is handled by the MusicController class and controlledthrough the TetrisLauncher and SettingViewController classes.

The playMusic method of the MusicController is called to start playing the music when the application is running. It loads the music file (/au/theme.mp3) and sets up a MediaPlayer to play the music indefinitely.

```java
public void playMusic() { 1 usage
    Media media = new Media(getClass().getResource( name: "/au/theme.mp3").toExternalForm());
    mediaPlayer = new MediaPlayer(media);
    mediaPlayer.setVolume(volume);
    mediaPlayer.setCycleCount(MediaPlayer.INDEFINITE);
    mediaPlayer.play();
}
```

b. Volume Control

The setMusicVolume method in the TetrisLauncher class allows changing the music volume. This method calls the setMusicVolume method of the MusicController. It method adjusts the volume of the MediaPlayer. If the volume is set to 0, the music is muted.

```java
public void setMusicVolume(double volume) { 1 usage
    this.volume = volume;
    if(volume == 0) {
        mediaPlayer.setVolume(0);
    } else {
        mediaPlayer.setVolume(volume);
    }
}
```

User Interaction:

In the SettingViewController class, the onMusicSliderChange method handles changes to the music volume slider. When the user adjusts the musicSlider,

the onMusicSliderChange method updates the music icon and calls the setMusicVolume method of the TetrisLauncher to change the music volume accordingly.

```java
public void onMusicSliderChange(MouseEvent mouseEvent) {  1 usage
    if(musicSlider.getValue() == 0) {
        musicIcon.setImage(new Image(getClass().getResource( name: "/assets/icon/music-off.png").toExternalForm()));
    } else {
        musicIcon.setImage(new Image(getClass().getResource( name: "/assets/icon/music-setting.png").toExternalForm()));
    }

    TetrisLauncher.setMusicVolume(musicSlider.getValue());
}
```

c. Tetrmino

The Tetrimino class represents a Tetris piece in the game. Each Tetrimino has multiple rotation states, a position on the grid, and a specific color. The class provides methods to manipulate the Tetrimino's position and rotation.

SHAPES: A 4D array defining the shapes of all Tetriminos in all their rotation states. Each Tetrimino type has four rotation states.

```
public static final int[][][][] SHAPES = {   2 usages
    { // I
            {{0, 1}, {1, 1}, {2, 1}, {3, 1}}, // 0°
            {{1, 0}, {1, 1}, {1, 2}, {1, 3}}, // 90°
            {{0, 1}, {1, 1}, {2, 1}, {3, 1}}, // 180°
            {{1, 0}, {1, 1}, {1, 2}, {1, 3}}  // 270°
    },
    { // J
            {{0, 0}, {0, 1}, {1, 1}, {2, 1}}, // 0°
            {{1, 0}, {2, 0}, {1, 1}, {1, 2}}, // 90°
            {{0, 1}, {1, 1}, {2, 1}, {2, 2}}, // 180°
            {{1, 0}, {1, 1}, {0, 2}, {1, 2}}  // 270°
    },
    { // L
            {{2, 0}, {0, 1}, {1, 1}, {2, 1}}, // 0°
            {{1, 0}, {1, 1}, {1, 2}, {2, 2}}, // 90°
            {{0, 1}, {1, 1}, {2, 1}, {0, 2}}, // 180°
            {{0, 0}, {1, 0}, {1, 1}, {1, 2}}  // 270°
    },
    //.............
```

COLORS: An array of Color objects representing the default colors for different types of Tetriminos

```
public static final Color[] COLORS = {
    Color.web("#00FFFF"), Color.web("#488BD4"),
    Color.web("#00FF00"), Color.web("#FF7F00"),
    Color.web("#FFFF00"), Color.web("#C200C2"),
    Color.web("#D41E3C")
};
```

Methods

```
public int[][] getBlocks() {...}

public int[][] getRelativeBlocks() { return rotations[rotationState]; }

public Color getColor() { return color; }

public void move(int dx, int dy) {...}

public void rotate() { rotationState = (rotationState + 1) % rotations.length; }

public void rotateBack() { rotationState = (rotationState - 1 + rotations.length) % rotations.length; }
```

- getBlocks(): Returns the absolute positions of the Tetrimino's blocks on the grid based on the current rotation state and position.
- getRelativeBlocks(): Returns the relative positions of the Tetrimino's blocks based on the current rotation state.
- getColor(): Returns the color of the Tetrimino.
- move(int dx, int dy): Moves the Tetrimino by (dx, dy) units on the grid.
- rotate(): Rotates the Tetrimino to the next rotation state.
- rotateBack(): Rotates the Tetrimino to the previous rotation state.

d. Tetris Game

The TetrisGame class manages the core logic of the Tetris game. It handles the game grid, the current Tetrimino, scoring, and rendering. The class provides methods to manipulate the Tetrimino, check for collisions, clear full rows, and render the game state.

Methods

```java
public Tetrimino createRandomTetrimino() { ···

public Tetrimino getCurrentTetrimino() { ···

public void setCurrentTetrimino(Tetrimino tetrimino) { ···

private boolean canMoveTo(int[][] blocks) { ···

public boolean moveTetrimino(int dx, int dy) { ···

public void rotateTetrimino() { ···

public void lockTetrimino() throws Exception { ···

private void clearFullRows() throws Exception { ···

private void clearRow(int row) { ···

public static int getScore() { ···

public void render() { ···

public int getScorePlus() { ···

public static void setScorePlus(int scorePlus) { ···
```

createRandomTetrimino(): Creates and returns a new random Tetrimino at the top of the grid.

getCurrentTetrimino(): Returns the current Tetrimino.

setCurrentTetrimino(Tetrimino tetrimino): Sets the current Tetrimino.

canMoveTo(int[][] blocks): Checks if the Tetrimino can move to the specified positions without colliding with the grid boundaries or occupied cells.

moveTetrimino(int dx, int dy): Moves the current Tetrimino by the specified amounts. If the move is not possible, it reverts the move.

rotateTetrimino(): Rotates the current Tetrimino. If the rotation causes a collision, it tries small adjustments to make the rotation valid. If no valid position is found, it reverts the rotation.

lockTetrimino(): Locks the current Tetrimino in place on the grid and checks for game over conditions. It also clears full rows and updates the score.

clearFullRows(): Checks for and clears full rows on the grid. It shifts the rows above down and updates the score. If the score reaches 3000, the game is won.

clearRow(int row): Clears the specified row and shifts the rows above down.

getScore(): Returns the current score.

render(): Renders the game grid and the current Tetrimino on the canvas.

getScorePlus(): Returns the score increment for clearing a row.

setScorePlus(int scorePlus): Sets the score increment for clearing a row.'

e. Choose Level

This function allows the user to choose the difficulty level of the game.

The ChooseLevelViewController provides buttons for selecting the difficulty level of the game. Each button click is handled by a corresponding method:

(clickEasyBtnHandler, clickNormalBtnHandler, clickHardBtnHandler), which calls the ViewController.getPlayingView method with the appropriate difficulty level as an argument.
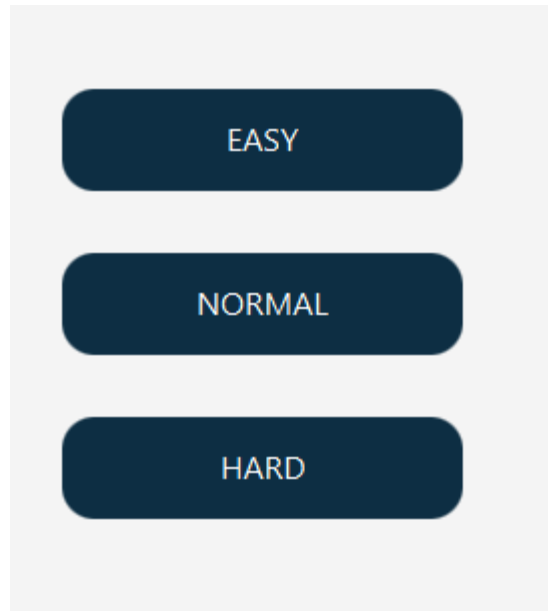
```
@FXML    1 usage
public void clickEasyBtnHandler() throws Exception {
    ViewController.getPlayingView( level: "easy");
}

@FXML    1 usage
public void clickNormalBtnHandler() throws Exception {
    ViewController.getPlayingView( level: "normal");
}

@FXML void clickHardBtnHandler() throws Exception {
    ViewController.getPlayingView( level: "hard");
}
```

f. Renderring Playing View

The PlayingViewController class is responsible for rendering the gameplay in our Tetris game. It displays the game state and handles the rendering of the game grid, current Tetrimino, and upcoming Tetriminos.

- **Canvas Components:**

gameCanvas: The main canvas where the game grid and current Tetrimino are rendered.

currentPlayCanvas: A canvas for rendering the current Tetrimino in a smaller view.

nextCanvas1, nextCanvas2, nextCanvas3: Canvases for rendering the next Tetriminos.

GraphicsContext: Each canvas has a GraphicsContext that is used for drawing shapes and images.

- **render() method**

The render() method in the TetrisGame class is responsible for drawing the game grid and the current Tetrimino on the gameCanvas

The gameLoop runs the game state, the gameLoop calls the render() method to refresh the display. This ensures that the visual representation of the game is always up-to-date with the current game state.

When player has a keypress, the state of the tetrimino will be update and render() method will be called instantly to present tetrimino on the screen

Example
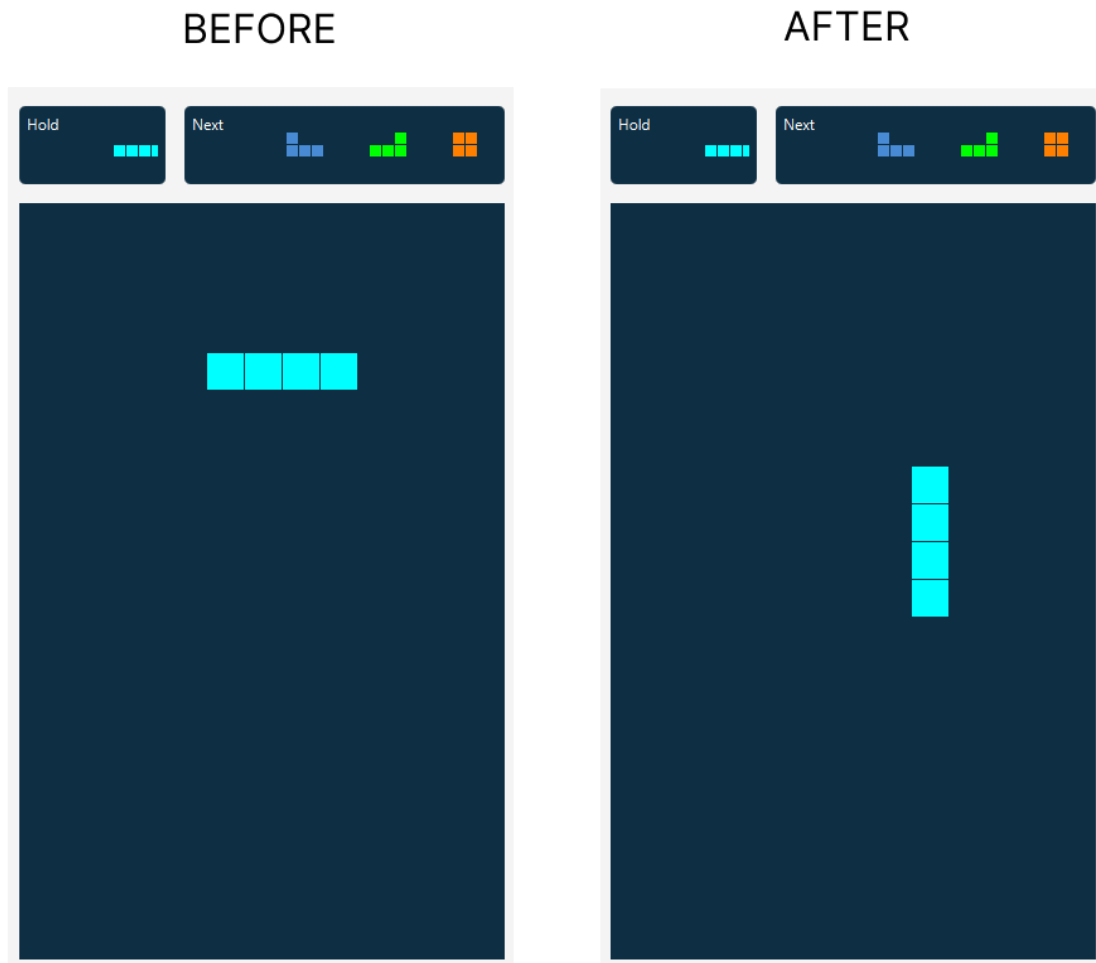
After the interval which is 0,5 second the game loop will move the tetrimino down by one block and call the render() method to present it on screen.

In this case, the UP key is pressed to update the state of rotation. The render() method will be called instantly to present it.
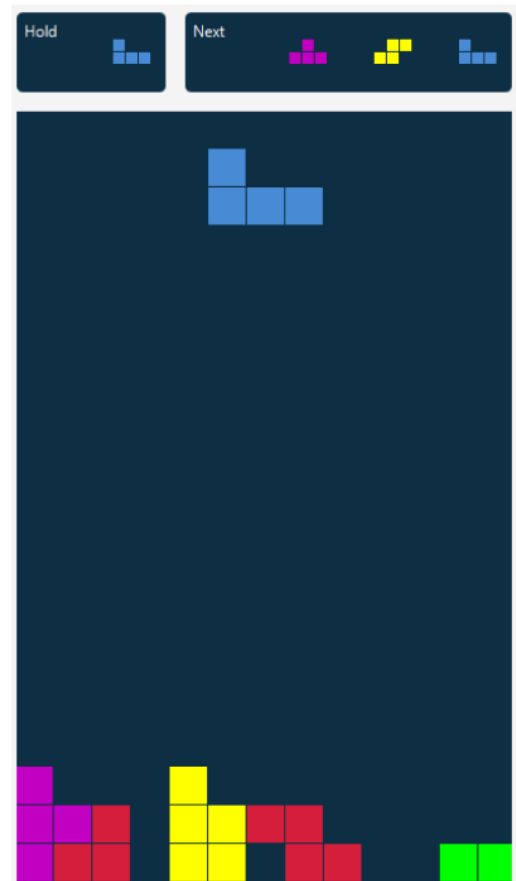
- **Clear row:**

  Our tetris game class has a method clearFullRows(), it will check through each row whether it is full or not. If it is full, the method clearRow() will be call to clear that specific row by updating the grid. And render() will be called to present a new grid on a screen.

  Example:
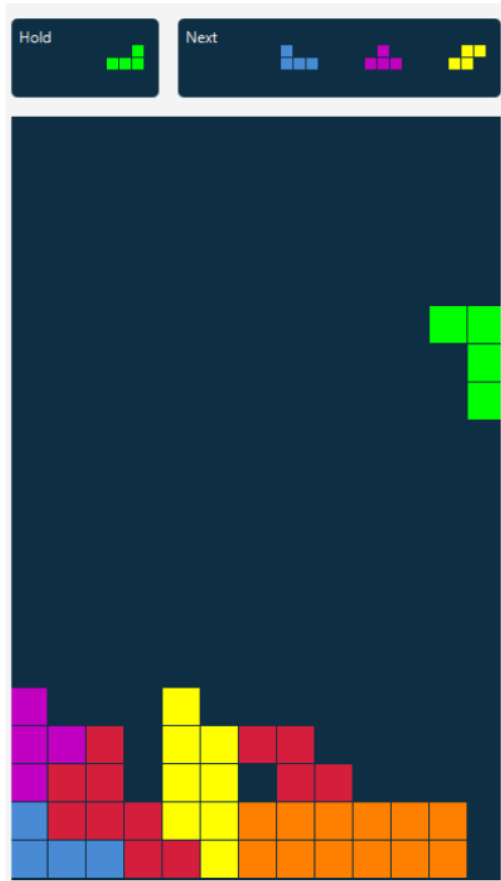
BEFORE   AFTER

- **Rendering current and next tetrimino:**
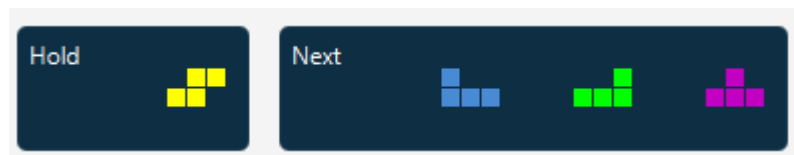
In our project, we have a place to present what the tetrimino the player currently has, and the next tetriminos they will occupy.

So, to do this, we create a list to store information about the next tetrimino, so whenever the block is locked, we just need to set the current tetrimino by the first one in the list, then remove it and we add a new random to the end at the list.

```
tetrisGame.setCurrentTetrimino(nextTetriminos.remove( index: 0));
nextTetriminos.add(tetrisGame.createRandomTetrimino());
```

```java
private void renderNextTetriminos() {  3 usages
    GraphicsContext[] nextCanvases = {
            nextCanvas1.getGraphicsContext2D(),
            nextCanvas2.getGraphicsContext2D(),
            nextCanvas3.getGraphicsContext2D()
    };

    // Xóa các canvas trước đó
    for (GraphicsContext gc : nextCanvases) {
        gc.clearRect( v: 0,  v1: 0, nextCanvas1.getWidth(), nextCanvas1.getHeight());
    }

    // Vẽ các khối tiếp theo trong danh sách Next
    for (int i = 0; i < nextTetriminos.size() && i < 3; i++) {
        drawTetrimino(nextTetriminos.get(i), nextCanvases[i],  offsetX: 16,  offsetY: 16);
    }
}

private void renderCurrentPlay() {  3 usages
    GraphicsContext gc = currentPlayCanvas.getGraphicsContext2D();
    gc.clearRect( v: 0,  v1: 0, currentPlayCanvas.getWidth(), currentPlayCanvas.getHeight());
    if (tetrisGame.getCurrentTetrimino() != null) {
        drawTetrimino(tetrisGame.getCurrentTetrimino(), gc,  offsetX: 16,  offsetY: 16);
    }
}

private void drawTetrimino(Tetrimino tetrimino, GraphicsContext gc, double offsetX, double offsetY) {  2 usages
    gc.clearRect( v: 0,  v1: 0, currentPlayCanvas.getWidth(), currentPlayCanvas.getHeight());
    gc.setFill(tetrimino.getColor());
    for (int[] block : tetrimino.getRelativeBlocks()) {
        gc.fillRect( v: (block[0] * 10 + offsetX) - 1,  v1: (block[1] * 10 + offsetY) - 1,  v2: 10 - 1,  v3: 10 - 1);
    }
}
```

- The renderCurrentPlay() method is responsible for rendering the current play Tetrimino on the Hold place.
- The renderNextTetriminos() method is responsible for rendering the next Tetriminos on the Next place.

g. Pause and Reload Game

To make players have a better experience, we have add some useful feature like pause and reload game
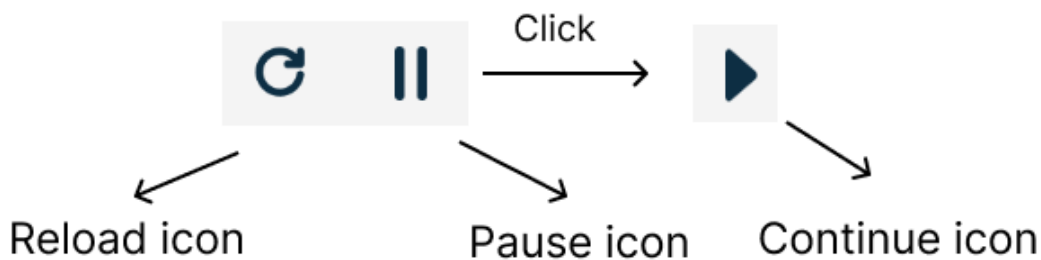
When the game is paused, the gameLoop is stopped, and the action icon is changed to the continue icon. That mean you can't move or rotate the tetrimino anymore until you click the continue icon.

When you click the reload icon, it will restarts the game by reinitializing the TetrisGame instance and resetting the game state. It stops the gameLoop, clears the nextTetriminos list and all the tetriminos that were blocked before..
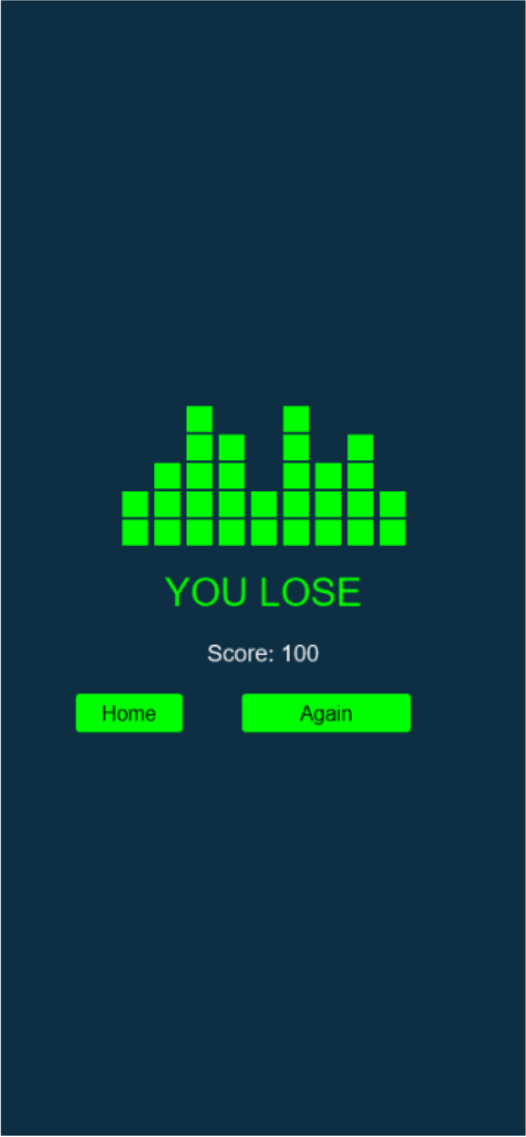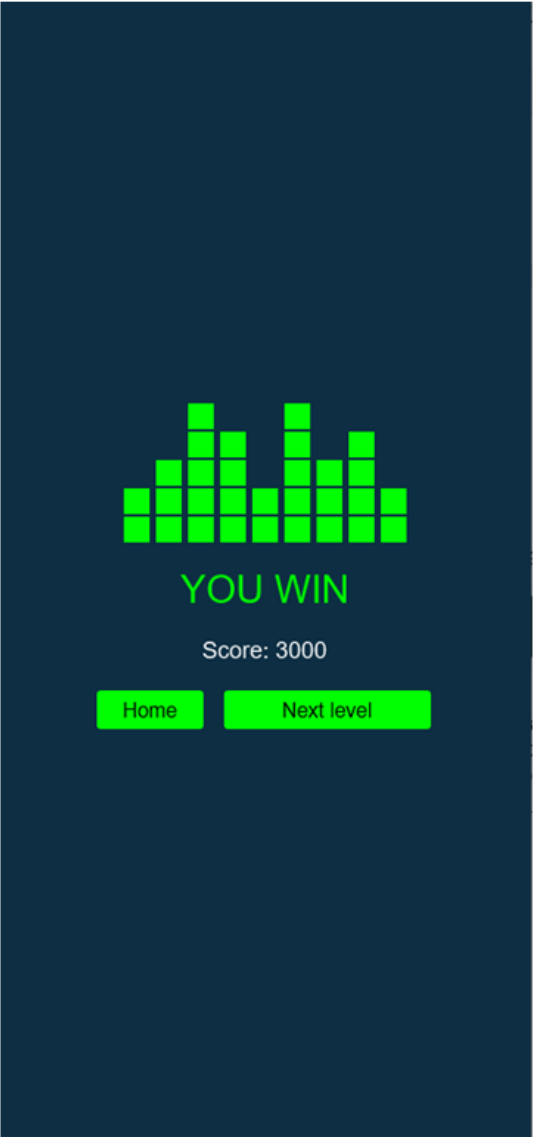
h. Finish

In our Tetris Game, we have a place to store player score
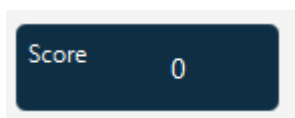
It will be zero at the beginning, and will be increased base on the rows that the player've cleared, the score for each row is different according to the level player choosing(The harder the level, the less points)

When player reach the score of 3000, they will win the game, otherwise they lose.

YOU WIN

Score: 3000

Home   Next level

YOU LOSE

Score: 100

Home   Again

CHAPTER 4: FINAL APP GAME

4.1 Source code (link Github):

https://github.com/hwHoai/OOP-Tetris-Game.git
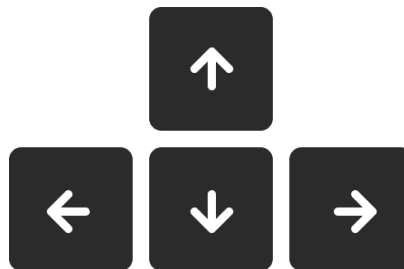
4.2 Demo video

https://github.com/anlecute/Video-Demo-Tetris.git

4.3 How to play

a. **Objective:** The objective of Tetris is to arrange falling shapes (called "tetrominoes") to complete horizontal lines. When a line is filled, it will disappear, and you'll earn points. The game ends when the shapes stack up and there's no room left to place new ones.

b. **Controls:**

- **Left Arrow (←):** Move the tetromino left.

- **Right Arrow (→):** Move the tetromino right.

- **Up Arrow (↑):** Rotate the tetromino 90 degrees clockwise.

- **Down Arrow (↓):** Speed up the falling of the tetromino (double the falling speed).

c. How to Play:

- At the start, tetrominoes will appear from the top of the screen and start falling down.

- You can use the controls to move and rotate the tetrominoes to fit them together and complete lines.

- When a line is completely filled, it will disappear, and you'll earn points.

d. Tips:

- Try to clear multiple lines at once for a higher score.

- Be careful not to let the tetrominoes stack up too high, as the game will end when they reach the top of the screen.

CHAPTER 5: EXPERIENCE

- Working on this Tetris project has been a great learning experience for our team, Cow Bò. From the very beginning, we understood that creating a game involves much more than just writing code. It's about crafting an experience for the players, which includes designing intuitive mechanics, engaging visuals, and captivating sound. Every element of the game, from the falling Tetriminos to the smooth animations and background music, has to work together to create a seamless and enjoyable experience, we have come to understand that:

Code, stack, and conquer – Tetris made through logic

- Throughout the development process, we've not only applied the programming skills learned in class but also developed problem-solving and creative thinking skills as we tackled different challenges, bugs, and new features. This project has encouraged us to explore and learn more on our own, beyond what was taught in school, which is essential in the fast-evolving tech world.

- As we move closer to completing this Tetris game, we realize that the knowledge we've gained goes beyond coding; it's about understanding user experience, design, and how to make a product that's both functional and fun. Our team, Cow Bò, is committed to completing this game and plans to expand our projects into mobile apps that provide even more entertainment and value to users.