**INTERNATIONAL UNIVERSITY - VNU HCMC**

**2024 - 2025 SEMESTER 02**

**IT069IU - OBJECT-ORIENTED PROGRAMING DNAT GROUP FINAL PROJECT**

**TETRIS GAME**

This report presents our final project for the OOP course: a Tetris game. The project demonstrates key OOP principles like encapsulation, inheritance, and polymorphism. It includes features such as dynamic block generation, collision detection, and a scoring system, showcasing how OOP is used to build engaging and maintainable software.

Contents

**Chap 01: Introduction.........................................................................3**

**1.1 About our group:...................................................................... 3**

**1.2 About the game project:.......................................................... 3**

**1.3 Our Tetris game:....................................................................... 4**

**1.4 References:............................................................................... 4**

**Chap 02: Software Requirements......................................................5**

**2.1 Functional Requirements:....................................................... 5**

**2.2 Non-Functional Requirements................................................ 5**

**2.3 Technical Requirements.......................................................... 6**

**2.4 System Architecture................................................................ 6**

**2.5 Use Case Scenario................................................................... 6**

**2.6 UML diagram.............................................................................7**

**2.7 Solid Principle:**......................................................................... 7

**Chap 03: Design & Implementation................................................ 10**

**1. Tetris.java.................................................................................. 10**

**2. StatusPane.java........................................................................ 12**

**3. OptionPane.java....................................................................... 14**

**4. MyShape.java............................................................................18**

**5. Board.java................................................................................. 21**

**6. Application.java........................................................................ 31**

**Chap 04: Demo..................................................................................32**

Chap 01: Introduction

**1.1 About our group:**

Group 1 is an Object-Oriented Programming project group. We have 4 members, including

|  |  |  |  |
| --- | --- | --- | --- |
| **Full name - GitHub username** | **Student ID** | **Contribute** | **Note** |
|  |  | 100% | Mino\_Bar.java + Mino\_L1.java + Mino\_L2.java + Mino\_Square.java + Mino\_T.java + Mino\_Z1.java + Mino\_Z2.java |
|  |  | 100% | Block.java + Mino.java + GameFrame.java + MenuScreen.java + Main.java UML |
|  |  | 100% | MyShape.java + UML model + Report |
|  |  | 100% | StatusPane.java + UML model + Report |

**1.2 About the game project:**

Tetris is one of the most iconic and enduring puzzle games in video game history. Created by Alexey Pajitnov in 1984, the game challenges players to manipulate falling geometric shapes, known as Tetrominoes, to form complete horizontal lines on a game board. When a line is successfully formed, it disappears, and the player earns points. As the game progresses, the falling blocks increase in speed, adding to the difficulty and excitement.

This report explores the key aspects of Tetris, including its gameplay mechanics, design, and impact on gaming culture. Additionally, the project demonstrates the

implementation of Tetris using modern programming techniques, highlighting challenges faced and solutions applied during the development process.

**1.3 Our Tetris game:**

As part of the ongoing development of our game project, we have introduced several new features aimed at enhancing the gameplay experience.

So first of all, we added some rules for our tetris game, that are

# 1. Objective:

- Arrange falling tetrominoes to create horizontal lines without gaps. When a line is completed, it disappears, and you earn points.

# 2. Controls:

- Move the blocks left or right, rotate the blocks to fit better and drop blocks faster by using the arrows keyboard.

# 3. Game area:

- The playfield is a grid, usually 10 columns wide and 20 rows tall.

# 4. Tetromino shape:

- Besides some basic shapes (such as: I, O, T, S, Z, J, L), we would like to add some special shapes.

# 5. Clearing Lines:

- Single line: 1 line cleared at once.

- Double, triple, or Tetris (4 lines) clears score higher points.

# 6. Game over:

- The game ends when blocks stack to the top of the playfield, and no more moves are possible.

# 7. Increasing Difficulty:

- As you play, the speed of falling blocks increases, making it harder to position them.

**1.4 References:**

Here are some key references that we use for our game project:

[https://viblo.asia/p/cung-thu-viet-mot-game-xep-hinh-tetris-hoan-chinh-tu-con-so-](https://viblo.asia/p/cung-thu-viet-mot-game-xep-hinh-tetris-hoan-chinh-tu-con-so-0-phan-1-giao-dien-va-game-loop-gDVK2mdA5Lj)

[0-phan-1-giao-dien-va-game-loop-gDVK2mdA5Lj](https://viblo.asia/p/cung-thu-viet-mot-game-xep-hinh-tetris-hoan-chinh-tu-con-so-0-phan-1-giao-dien-va-game-loop-gDVK2mdA5Lj)

Chap 02:

Software Requirements

## 2.1 Functional Requirements

### 1. Game Mechanics

* The game must spawn random Tetrominoes (Tetris shapes) at the top of the game grid.
* Players can move Tetrominoes left, right, down, or rotate them.
* Completed rows should be cleared, and the score should increase accordingly.

### 2. Level Progression

* The game speed should increase as the player completes more rows.

### 3. Score System

* Players are awarded points for clearing rows.
* Clearing multiple rows at once yields higher scores.

### 4. Game Over

* The game ends when a Tetromino cannot fit at the top of the grid.

### 5. Pause/Restart

* The game must include options to pause or restart via user input.

### 6. Next Piece Preview

* The game displays a preview of the next Tetromino to allow players to plan ahead.

### 7. Hold Function (Optional/Extension)

* Players can press a specific key (e.g., 'C') to hold the current Tetromino and swap it with the next one.
* The hold function can be used only once per drop.

### 8. High Score Tracking

* The game saves and displays the highest score achieved in previous sessions, either via file or in memory (optional feature).

### 9. Sound Effects

* The game plays sound effects for key actions (e.g., row clear, hard drop, game over, button clicks).

### 10. Difficulty Selection

* Players can choose the difficulty (e.g., Easy, Normal, Hard, Extreme) before starting the game.
* Each difficulty level determines the initial speed, possible Tetromino types, and scoring modifiers.

### 11. User Interface Navigation

* The main menu allows users to:  
  + Start a new game
  + Adjust options (such as sound or keybindings)
  + View high scores
  + Exit the game

### 12. Soft Drop and Hard Drop

* **Soft Drop:** Holding the down arrow increases the falling speed of the Tetromino.
* **Hard Drop:** Pressing the spacebar instantly drops the Tetromino to the lowest available position.

### 13. Game Resume

* If the game is paused, resuming will restore the state exactly as it was left, with no loss of progress.

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 2.2 Non-Functional Requirements

### 1. Usability

* Controls must be intuitive and responsive. The game uses keyboard input (arrow keys, P for pause, etc.).

### 2. Scalability

* The code is modular and object-oriented, making it easy to extend features (e.g., new Tetromino shapes or effects).

### 3. Cross-Platform

* Built with Java and Swing, the game is playable on any system with a Java runtime environment (Windows, macOS, Linux).

### 4. Performance

* The game loop should update and render smoothly at a consistent frame rate to ensure responsive gameplay, even on lower-end hardware.

### 5. Accessibility

* Font sizes, contrast, and key mappings should be adjustable to accommodate players with different needs.

### 6. Reliability

* The game should not crash or lose state during normal operation.
* Error handling must be implemented for all file I/O (such as high score saving/loading).

### 7. Documentation

* The codebase should include comments and a README file explaining setup, controls, and contribution guidelines.

## 

## 2.3 Technical Requirements

### 1. Programming Language

* Java

### 2. Development Environment

* IntelliJ IDEA, Eclipse, Visual Studio Code, or any Java-supporting IDE

### 3. Libraries

* Java AWT and Swing for graphics and UI rendering.
* Java event-handling framework for keyboard input and interaction.

### 4. Design Pattern / OOP Principles

* **Inheritance:** All Tetromino types inherit from a base Mino class.
* **Encapsulation:** Game logic and data (e.g., PlayManager, Grid, ScoreTracker) are encapsulated in separate classes.
* **Polymorphism:** Tetromino behaviors (movement, rotation) are defined polymorphically across Tetromino subclasses.

### 5. Resource Management

* All resources such as images and sound files should be loaded efficiently and managed to avoid memory leaks.

### 6. Version Control

* The project must be managed under version control (e.g., Git), with a clear branching strategy for new features and bug fixes.

### 7. Testing

* Basic unit tests or manual test cases should be defined, especially for scoring, row clearing, and Tetromino rotation logic.

## 2.4 System Architecture

### 1. Class Design

* **GamePanel:** The main game panel; controls initialization, repainting, and the game loop.
* **PlayManager:** Manages core gameplay including spawning Tetrominoes, detecting game over, clearing rows, and tracking score and level.
* **Mino and Subclasses:** Represent different Tetromino types (Mino\_Bar, Mino\_L1, Mino\_T, etc.) with rotation and movement logic.
* **Block:** Represents individual squares that make up a Tetromino.
* **KeyHandler:** Handles user input such as arrow keys, pause (P), fast fall (S), and hard drop (spacebar).
* **Sound:** Handles sound effects during game events.
* **ScoreTracker (conceptual):** Score and line counters are currently managed inside PlayManager.

### 2. Graphics and UI

* The game uses 2D rendering to display the grid, Tetrominoes, and UI elements like score, level, and next piece preview.

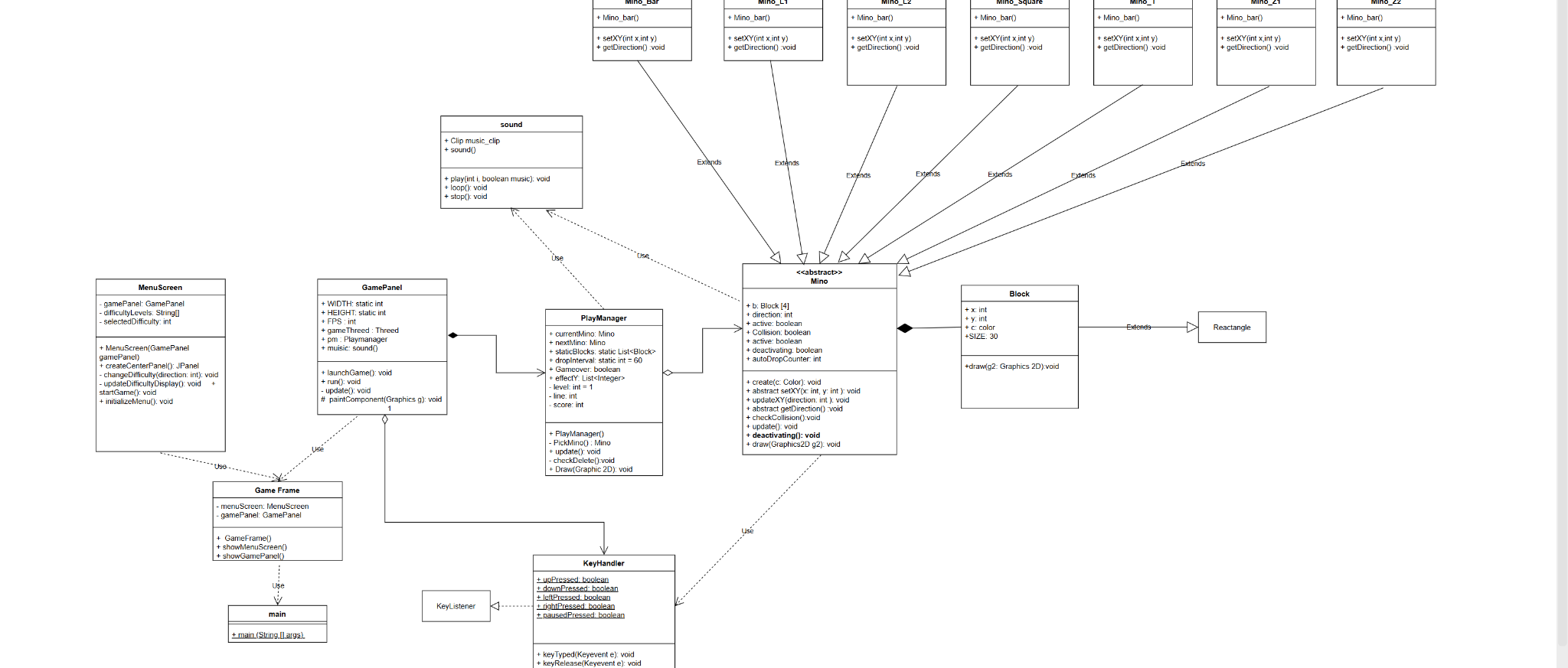
### 3. New Features

* A slider or key to adjust falling speed.
* Difficulty selection (Normal, Hard, Extreme) affecting speed and Tetromino types.
* High score and deleted lines are tracked and displayed.
* Down arrow: soft drop (temporarily increase falling speed).
* P key: pause the game.

**2.5 Use Case Scenario**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Use Case Name** | **Actor** | **Preconditions** | **Main Flow** |  | **Postconditions** |
|  | Start Game | Player | Game is installed and closed | 1. Player opens the game.  2. System loads the main menu.  3. Player clicks "Start".  4. System initializes the board and begins a new game session. |  | Game session is started. |
|  | Pause Game | Player | Game is currently running | 1. Player presses the "Pause" button.2. System freezes the game state and displays the pause menu. |  | Game is paused. |
|  | Change Difficulty | Player | Main menu is open | 1. Player selects "Difficulty".  2. System displays difficulty options.  3. Player selects a difficulty.  4. System updates the setting. |  | Difficulty level is updated. |
|  | Exit Game | Player | Game is running or in menu | 1. Player selects "Exit".  2. System asks for confirmation.  3. Player confirms.  4. System closes the game. |  | Game is closed. |
|  | Play Sound Effect | System | Event is triggered (e.g., line cleared) | 1. A game event occurs (e.g., row is cleared).  2. System plays the corresponding sound effect. |  | Sound effect is played. |

**2.6 UML diagram**

****

**2.7 Solid Principle:**

**1. Single Responsibility Principle (SRP) Application.java**

- Purpose: Unclear due to lack of methods; likely acts as a container or utility class.

- Assessment: Requires further inspection to confirm adherence to SRP.

**Board.java**

- Purpose: Core game logic and board management.

- Assessment: Violates SRP if it handles too many unrelated concerns (e.g., game state, rendering, and user input).

**MyShape.java**

- Purpose: Represents and manipulates Tetris shapes.

- Assessment: Compliant with SRP as it focuses on shape-related functionality.

**OptionPane.java**

- Purpose: Manages user interactions through dialog boxes.

- Assessment: Compliant with SRP as it focuses solely on UI interactions.

**StatusPane.java**

- Purpose: Tracks and updates game status (e.g., score, high score).

- Assessment: Compliant with SRP as it focuses on status updates.

**Tetris.java**

- Purpose: Initializes and manages the game window and main menu.

- Assessment: Compliant with SRP if it limits itself to UI initialization and event handling.

**2. Open-Closed Principle (OCP) Board.java**

- Assessment: Could violate OCP if changes to game logic require modification of the class rather than extending it.

**MyShape.java**

- Assessment: Compliant with OCP if new shapes can be added without altering existing methods.

**OptionPane.java** and **StatusPane.java**

- Assessment: Likely compliant with OCP if they allow customization through inheritance or configuration.

**Tetris.java**

- Assessment: May violate OCP if modifications to UI require changing the class directly.

**3. Liskov Substitution Principle (LSP)**

- Subtypes must be substitutable for their base types without altering program behavior.

- No explicit inheritance hierarchies were observed, but further inspection is needed to confirm compliance.

**4. Interface Segregation Principle (ISP)**

- Classes should not be forced to implement interfaces they do not use.

**Board.java** and **Tetris.java**

- Assessment: May violate ISP if they are required to implement unrelated methods due to broad interface contracts.

- Other classes seem focused, reducing the risk of ISP violations.

**5. Dependency Inversion Principle (DIP)**

- High-level modules should not depend on low-level modules; both should depend on abstractions.

**Board.java**

- Assessment: Could violate DIP if it directly depends on concrete classes like

MyShape or UI components.

**Tetris.java**

- Assessment: May violate DIP if it directly manages low-level UI components without abstractions.

Application.java Class: Application Methods: None listed.

**Board.java**

- Classes: Board, TAdapter

+ Methods: Extensive list, including game logic (e.g., addGameSpeed, start, pause, dropDown, newPiece, tryMove, etc.).

**MyShape.java**

- Class: MyShape

+ Methods: Related to handling shapes in the game (e.g., setShape, rotateLeft, rotateRight, setRandomShape, etc.).

**OptionPane.java**

- Class: OptionPane

+ Methods: User interaction handling (e.g., initializeComponents, actionPerformed, stateChanged).

**StatusPane.java**

- Class: StatusPane

+ Methods: Status tracking and updates (e.g., scored, highScored, Counted, etc.).

**Tetris.java**

- Classes: Tetris, MenuHandler

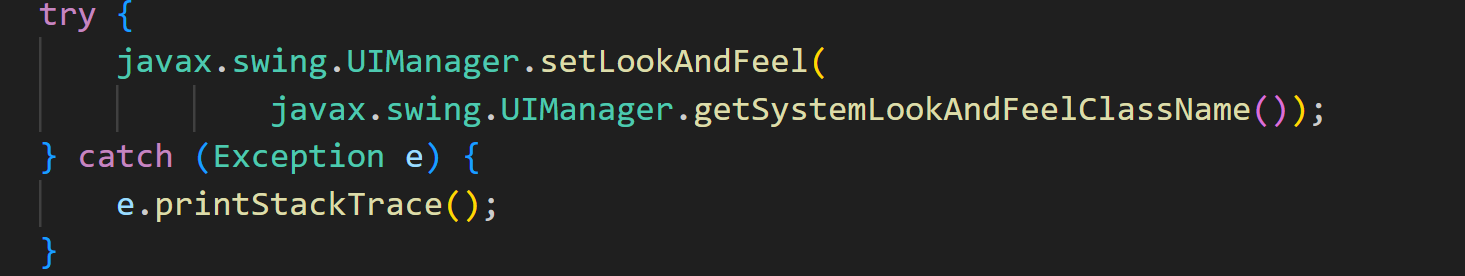
+ Methods: Game initialization and UI logic (e.g., initializeComponents, getStatusBar, actionPerformed).

**Chap 3: Implement**

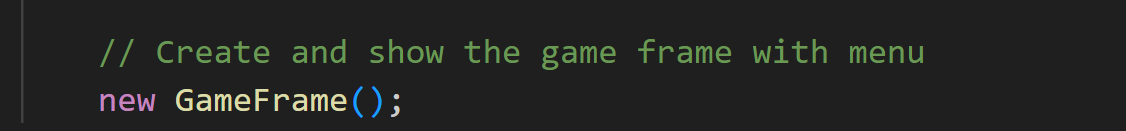
**Program Structure**

**1. Main.java**

* Set system look and feel for better appearance

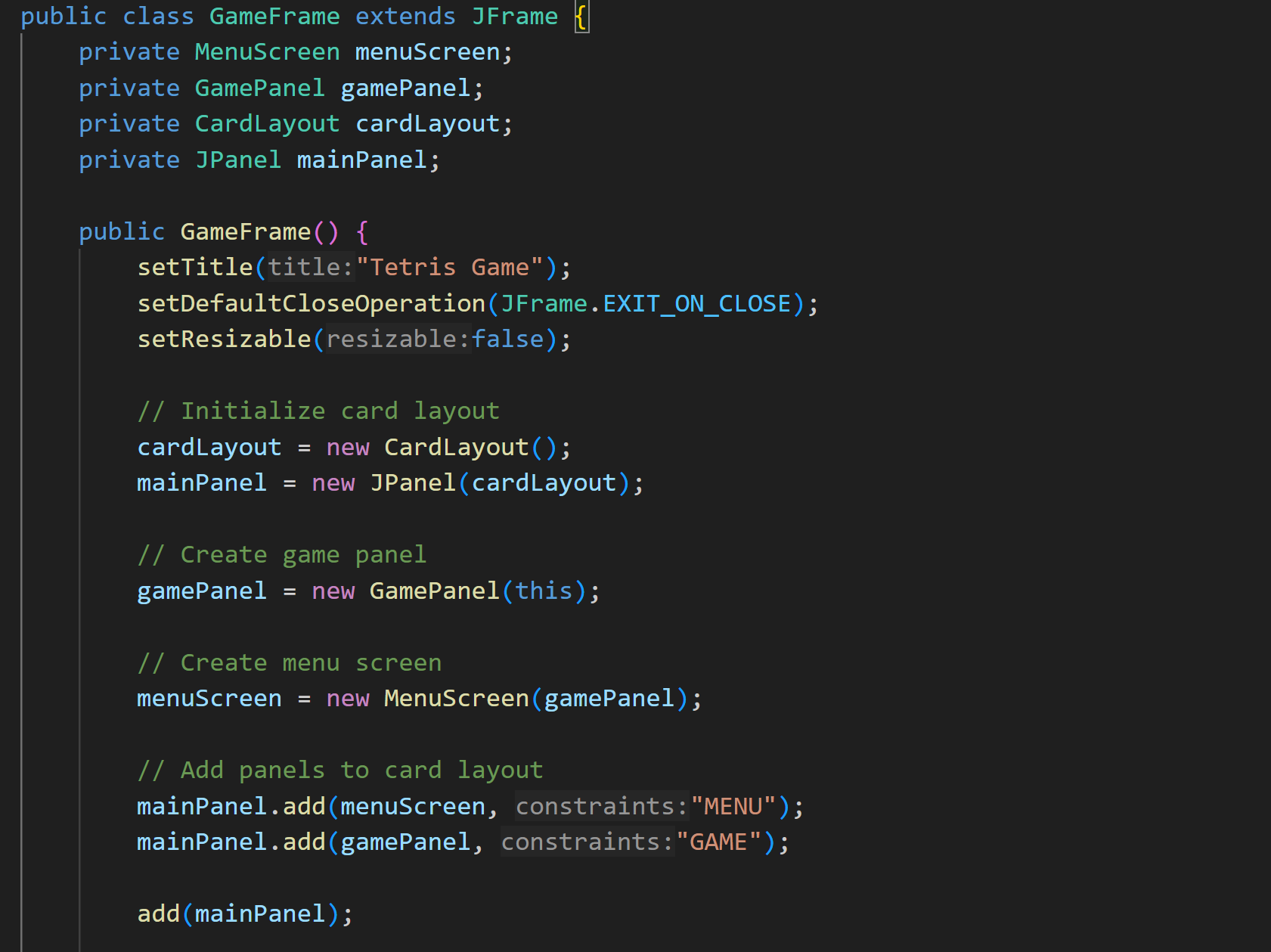


* Create and show the main game window:

****

**2.GameFrame.java**

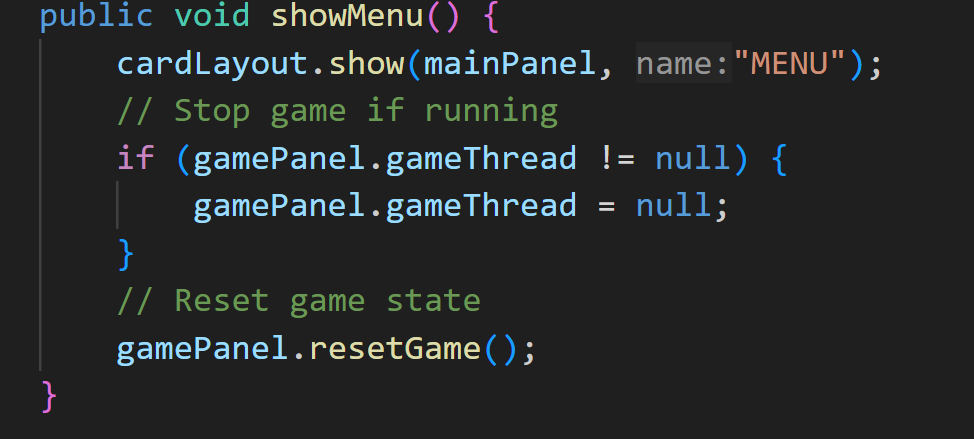
* Set up a card layout that holds both the menu and the game panel, allowing smooth screen switching.

****

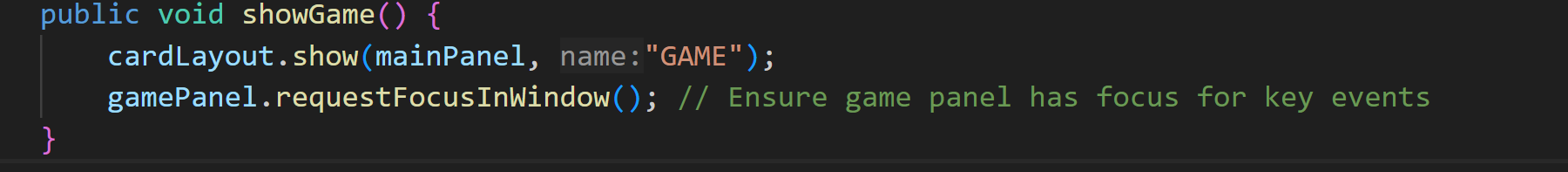
* show the main menu first when the game starts.

****

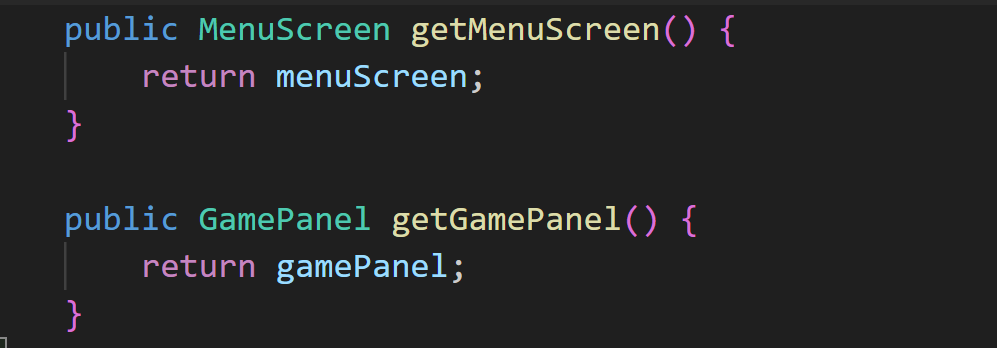
* Displays the menu, stops the game thread, and resets the game state.



* Displays the menu, stops the game thread, and resets the game state.

****

* getter method

****

**3. MenuScreen.java**

**4. GamePanel.java**

**5. KeyHandler.java**

**6. Sound.java**

**7. Mino.java**

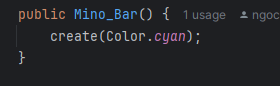
**8. Block.java**

**9. Mino (Bar, Z1, Z2, T, L1, L2, Square).java**

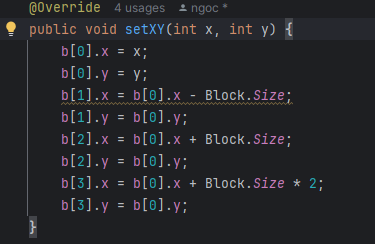
* The classes **Bar**, **Z1**, **Z2**, **T**, **L1**, **L2**, and **Square** inherit from the **Mino** class, which serves as the base class representing a block in the Tetris game.



* Each class represents a distinct block shape, such as the long bar (**Bar**), Z-shaped blocks (**Z1**, **Z2**), the T-block, L-shaped blocks (**L1**, **L2**), and the square block (**Square**).
* In the constructor of each class, the block is created with a unique color via the create(Color) method.



* Every class implements the setXY(int x, int y) method to define the initial coordinates of the four individual blocks that form the shape, based on a given origin position (x, y).



* The methods getDirection1(), getDirection2(), getDirection3(), and getDirection4() define the block's shape in different rotation states, where each class implements its unique rotation logic by computing temporary coordinates using tempB[], which are then applied through the updateXY() method.



* Some blocks, like **Square**, only have one orientation, so their rotation methods will be identical. Other blocks have either two or four rotation orientations, and their respective methods are implemented accordingly.

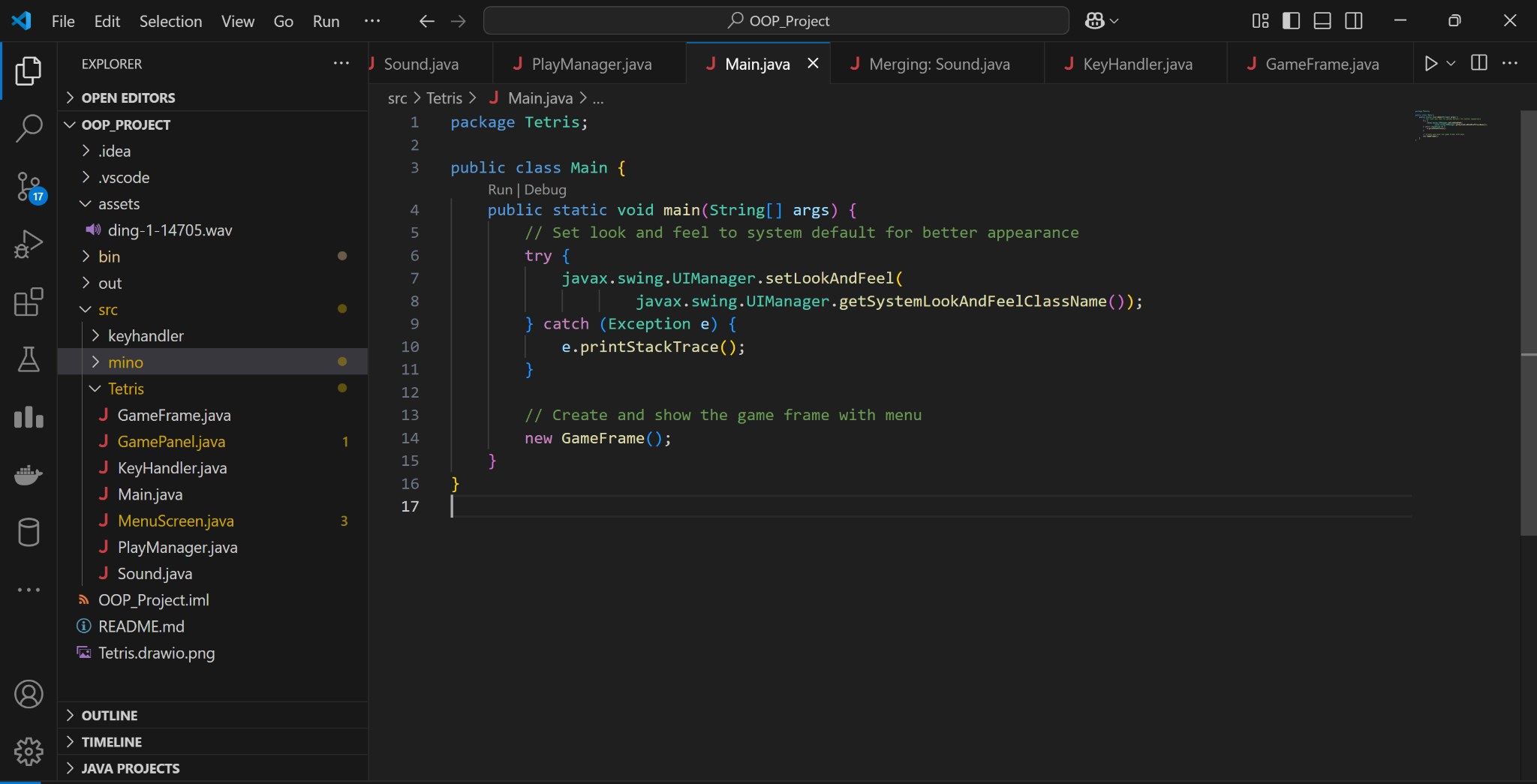
**Chap 04: Demo**

**Instruction**

**1. Open source code files with IDE (VSCode, NetBeans, etc..)**



**2. Choose Main.java and click Run**



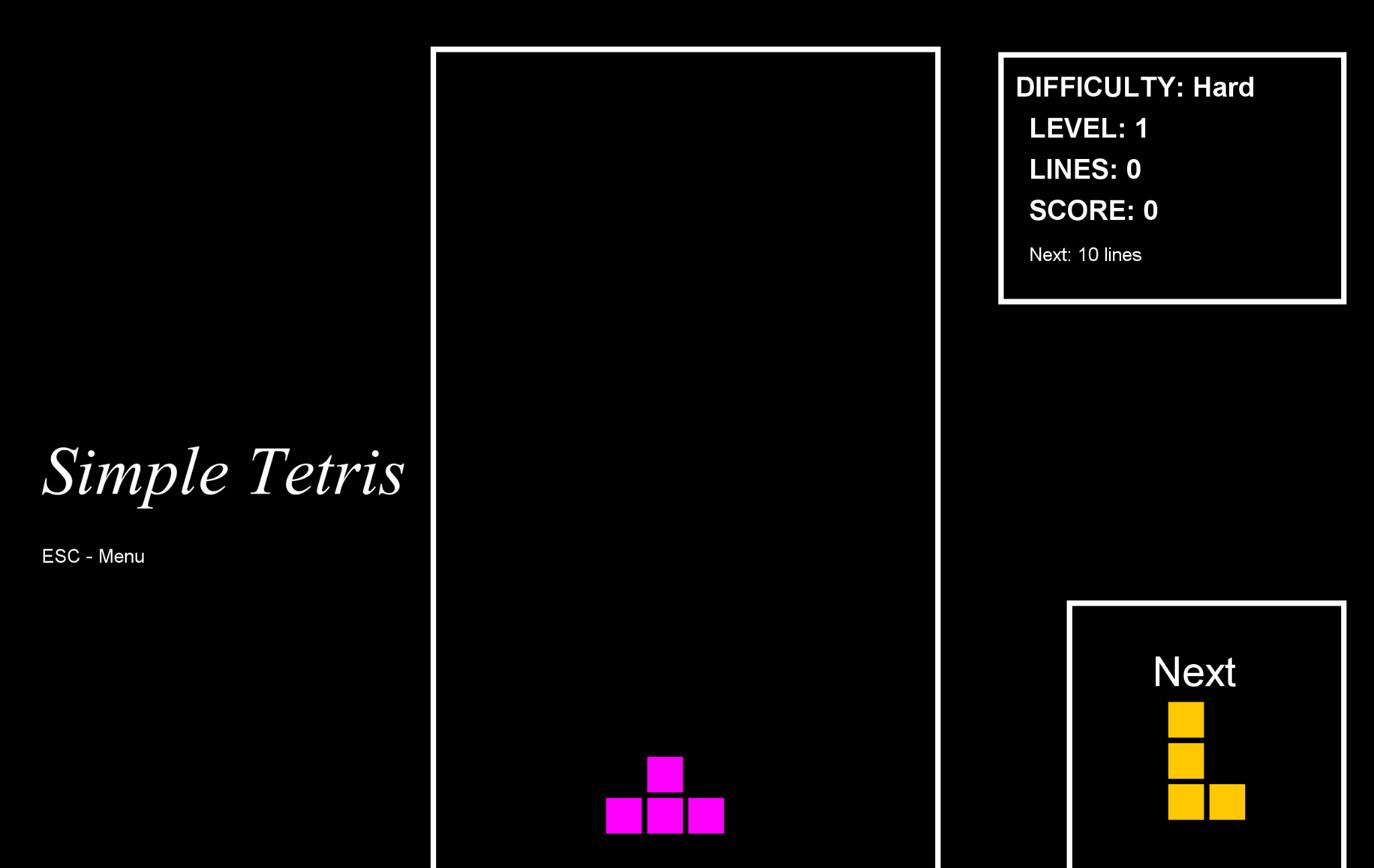
**3. A board game will appear on the screen**



**4. You can adjust difficulty**

****

**5. Click Start Game button to start a new game**

****

A new block appears and falls down.

You can keep pressing the ‘down’ button to make the block fall down faster or press Space to let the block fall immediately.

**7. A row is terminated when it is completed**



At the time, the system will update line number and score.

**11. Finally, you can try Expert mode, an upgrade version of Hard mode where fastest speed.**

****

37