***OOP Final Report***

***Tetris.java:***

**Tetris:**

The code written in Tetris represents a simplified version of the classic game Tetris. It is written in Java and uses the Swing library for GUI components. Let's go through the code and explain its functionality:

A screen shot of a computer program

Description automatically generated with low confidence

**J-Frame:**

The code begins with importing necessary packages and defining the class Tetris, which extends the J-Frame class. It represents the main game window and handles game logic and user input.



**Board Panel and Side Panel:**

The Tetris class has various member variables, including Board Panel and Side Panel instances for displaying the game board and side information, boolean variables to track game states (paused, game over, new game), score and level variables, a random number generator, and variables related to the current and next game pieces.

A screenshot of a computer

Description automatically generated with medium confidence

**Key Listener:**

The Tetris class constructor initializes the window properties, sets the layout, creates instances of Board Panel and Side Panel, and adds them to the window. It also adds a custom Key Lestener to handle user input.

A picture containing text, font, screenshot

Description automatically generated

The Tetris class contains methods to start the game, update the game logic, render the game on the screen, reset the game, and spawn new game pieces.



**Start game:**

The Start game () method is the main game loop. It continuously updates the game logic, handles user input, and renders the game on the screen. It also controls the game speed and frame rate.

A screen shot of a computer program

Description automatically generated with low confidence

**Update game:**

The Update game () method is responsible for updating the game state and checking for collision, clearing lines, increasing the score, and advancing the game to the next level.



**Render game:**

The Render game () method repaints the Board panel and Side panel to display the current game state.

A picture containing text, font, screenshot

Description automatically generated

**Restart game:**

The Restart game () method resets the game variables to their default values, including score, level, game speed, and current and next game pieces. It also clears the game board.

A picture containing text, font, screenshot, graphics

Description automatically generated

**Spawn piece:**

The Spawn piece () method is called to create a new game piece and initialize its position and rotation. It checks if the spawn point is valid, and if not, it sets the game over state.



**Rotate piece:**

The Rotate piece () method attempts to rotate the current game piece to a new rotation. It checks for valid positions and adjusts the piece's position if necessary to avoid clipping out of the game board.



**Remaining methods:**

The remaining methods are getter methods to check the game state (paused, game over, new game).





**Conclusion:**

So, the code sets up the Tetris game window, handles user input, updates the game logic, and renders the game on the screen. It provides the basic functionality to play the game, including moving and rotating pieces, clearing lines, keeping score, and advancing levels.

***BoardPanel.java:***

**Board Panel:**

The Java implementation of the graphical user interface (GUI) for a Tetris game. It consists of a class called Board Panel, which extends the J-panel class to create a custom panel for displaying the game grid and handling various aspects related to the game board.



The Board panel class contains several constants that define the dimensions, colors, and properties of the game board and its components. For example, it defines the size and colors of the tiles, the width of the shading on the tiles, the number of columns and rows on the board, and the fonts used for displaying text.

**Tiles:**

The class has instance variable Tiles, which is a 2D array representing the game board. Each element of the array corresponds to a tile on the board and can hold a Tile Type object that represents the type of tile (e.g., empty, filled, or different Tetris shapes).

A picture containing font, text, screenshot, graphics

Description automatically generated

**Clear:**

The class provides various methods to interact with the game board. The Clear method resets the board by setting all tiles to null. The “is” Valid and Empty method checks if a specific position on the board is valid and empty for placing a Tetris piece. It considers the boundaries of the board and checks for overlapping with existing tiles. The add piece method adds a Tetris piece to the board by setting the corresponding tiles based on the piece's type, position, and rotation.



**Check Lines:**

There are also methods related to game logic, such as Check Lines, which checks if any lines on the board are filled and clears them. It iterates through each row and checks if all tiles in that row are occupied. If a line is cleared, the method shifts the rows above it down by one to simulate the effect of clearing the line.

A picture containing text, font, screenshot, graphics

Description automatically generated

**Is Occupied, set tile, Get tile:**

The Is Occupied, set tile, and get tile methods provide low-level operations for accessing and modifying individual tiles on the board. They check if a specific tile is occupied by checking if it contains a non-null tile Type object, set a tile to a specific type, and retrieve the type of a tile, respectively.







**Paint Component:**

The Paint Component method is overridden to customize the rendering of the board. It first calls the superclass's Paint Component method to perform any default painting, and then proceeds to draw the board based on the current game state. If the game is paused, it displays a "PAUSED" message. If it's a new game or game over, it shows the corresponding messages. Otherwise, it renders the tiles on the board, the current piece, and a ghost piece that shows where the current piece will land.

A picture containing text, font, screenshot, line

Description automatically generated

**Graphics:**

The method uses Graphics object to draw various shapes and text on the panel. It translates the coordinate system to simplify positioning, and then loops through the tiles on the board, drawing each occupied tile using the draw Tile method. It also draws the background grid for visual appeal.



**Conclusion:**

The Board Panel class provides the graphical representation of the Tetris game board and handles the rendering and manipulation of tiles on the board. It encapsulates the logic for checking valid moves, adding, and clearing pieces, and updating the display based on the game state. The code demonstrates the use of Java's Swing library for creating GUI components and graphics rendering.

***Clock.java:***

**Clock:**  
Clock.java represents Clock class in Java that is designed to measure time and track elapsed cycles. This class is particularly useful in game development or any application that requires precise timing.



**Clock Class:**

The Clock class encapsulates several private instance variables.

Millis Per Cycle represents the number of milliseconds that make up one cycle.



The Last update stores the timestamp of the last update, allowing the calculation of delta time.



Elapsed Cycles keeps track of the number of cycles that have elapsed and have not yet been polled.

A picture containing font, graphics, screenshot, text

Description automatically generated

Excess Cycles holds the amount of excess time towards the next elapsed cycle.

A picture containing font, graphics, screenshot, text

Description automatically generated

Lastly, is paused is a boolean flag indicating whether the clock is currently paused.

A picture containing text, font, screenshot, graphics

Description automatically generated

**Clock:**

The constructor initializes the Clock object by accepting the desired number of cycles Per Second. It calculates and sets the Millis Per Cycle based on this value, and then calls the reset method to initialize other variables.

A picture containing text, font, screenshot

Description automatically generated

**Set Cycle Per Second:**

The set cycle per second method adjusts the Millis per Cycle based on the provided Cycles per Second value. It calculates the reciprocal of cyclesPerSecond, multiplies it by 1000 to convert it to milliseconds, and assigns the result to Millis per Cycle.



**Reset:**

The reset method resets the clock's statistics. It sets elapsed cycles and excess cycles to 0, captures the current time using the get current time method and assigns it to last update, and sets is paused too false.

A picture containing text, font, screenshot

Description automatically generated

**Update:**

The update method is responsible for updating the clock's statistics. It begins by obtaining the current time using get Current Time. It calculates the delta time by subtracting the last update timestamp from the current time and adds any excess cycles from the previous cycle. If the clock is not paused, it updates the elapsed cycles by dividing the delta time by Millis per Cycle and taking the floor value. The remaining fraction of the division is assigned to excess cycles. Finally, it updates the last update with the current time for the next cycle.

A picture containing text, screenshot, font

Description automatically generated

**Set Paused:**

The set paused method allows the clock to be paused or unpaused. It takes a Boolean parameter paused and assigns its value to is paused. When the clock is paused, the elapsed cycles and cycle excess will not be updated. However, the update method should still be called for regularly to ensure accurate timing calculations.



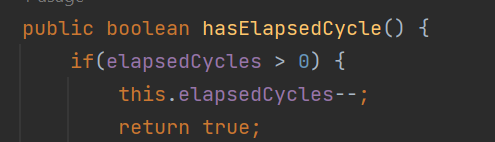
**Is paused:**

The is paused method simply returns the value of the is paused flag, indicating whether the clock is currently paused.



**Has Elapsed cycle:**

The has elapsed cycle method checks if there is an elapsed cycle. If elapsed cycles are greater than 0, it decreases lapsed cycles and returns true, indicating that a cycle has elapsed. Otherwise, it turns out false. This method allows the caller to determine if a cycle has completed and take appropriate actions.



**Peek Elapsed cycle:**

The peek elapsed cycles method provides a way to check if an elapsed cycle exists without modifying the elapsed cycles variable. It returns true if elapsed cycles are greater than 0, indicating that a cycle has elapsed, and false otherwise.



**Get current time:**

The get current time method is a private helper method that calculates the current time in milliseconds using the computer's high-resolution clock. It uses System.nano time () to obtain the current time in nanoseconds and then divides it by 1000000L to convert it to milliseconds.



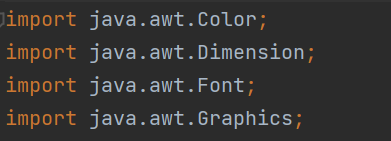
**Conclusion:**

The clock class provides functionality for measuring time and tracking elapsed cycles. It allows setting cycles per second, updating the clock, pausing/unpausing, and checking for elapsed cycles. This code is useful for applications where precise timing and synchronization are required, such as game development or real-time simulations.

***SidePanel.java:***

**Side Panel:**

Side panel class, which is a crucial component of the Tetris game user interface. It is responsible for displaying various information about the game, such as the next piece, score, current level, and controls. Understanding the code requires a detailed explanation of its structure and functionality.

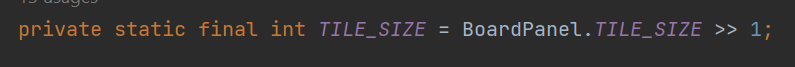


The Side panel class is defined as a subclass of J-panel, a Swing component used for creating containers to hold other components. By extending J-panel, the side panel class inherits the properties and behavior of a panel, allowing it to be added to a larger graphical user interface.

A picture containing text, font, screenshot, graphics

Description automatically generated

The side panel class begins with the declaration of several constants that determine the dimensions, positions, fonts, and colors used for drawing the game information on the panel. These constants provide a standardized layout and appearance for the panel, making it easier to maintain and modify the code in the future.





**Tetris:**

One of the class variables is an instance of the Tetris class, which represents the game instance associated with the side panel. The Tetris class likely contains game logic and state information. By having a reference to the Tetris instance, the side panel can retrieve information from the game, such as the level, score, and the next piece to be displayed.

The constructor of the Side panel class takes a Tetris instance as a parameter. This allows the Side panel to be associated with a specific game, ensuring that the displayed information is synchronized with the game state. In the constructor, the preferred size of the panel is set to ensure it has an appropriate size within the user interface, and the background color is set to black.

A picture containing text, screenshot, font

Description automatically generated

**Paint Component:**

The Side panel class overrides the Paint component method, which is a fundamental method in Swing for custom painting. This method is automatically called when the panel needs to be repainted, such as when it becomes visible or its contents change.

Within the paint component method, the code first calls super.paint component (g) to ensure proper painting behavior. This is a recommended practice when overriding paint component to avoid potential issues with painting artifacts.

A screen shot of a computer

Description automatically generated with low confidence

**Graphics:**

The method then uses the Graphics object g to draw various elements on the panel. The Graphics object provides methods for drawing shapes, images, and text on a component.

**Draw color:**

To establish a consistent visual style, the method sets the color for drawing to a predefined Draw color. This color will be used for drawing text and the preview box.



**Offset:**

The code utilizes the offset variable to keep track of the y-coordinate of each drawn string. This allows for flexible positioning of the elements on the panel. The offset variable is incremented as each string is drawn, ensuring proper spacing between the elements.



**Stats:**

The first element drawn is the "Stats" category. The code sets the font to a predefined Large Font and draws the string "Stats" at the specified Small Inset x-coordinate and the current Offset y-coordinate. The offset is then updated by adding Text Stride, which represents the number of pixels to offset between each string. The font is then set to a predefined Small Font, and the level and score values are drawn below the category title. The tetris.get Level () and tetris.get Score () methods retrieve the corresponding values from the Tetris instance. The offset is updated again after each string is drawn.

A picture containing text, screenshot, font

Description automatically generated

**Controls:**

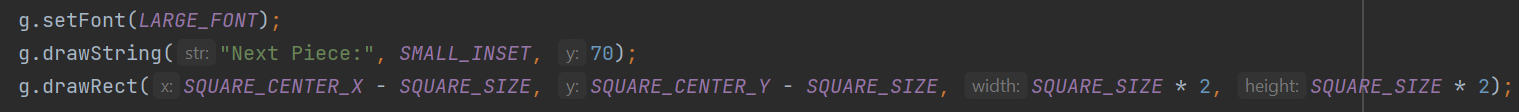
The code continues by drawing the "Controls" category. It follows a similar pattern to the "Stats" category, setting the font, drawing the category title, and then drawing the control instructions. The control instructions specify the keyboard inputs for moving, rotating, dropping, and pausing the game. Each control instruction is drawn using the predefined fonts and Large Inset x-coordinate, with the offset updated accordingly.

A picture containing text, screenshot, font, line

Description automatically generated

**Next piece preview box:**

Next, the code draws the next piece preview box. It starts by setting the font to Large Font and drawing the string "Next Piece:" at the specified Small Inset x-coordinate and a fixed y-coordinate of 70. The code then draws a rectangular outline representing the preview area. The position and dimensions of the outline are determined by the predefined constants, such as Square Center X, Square Center Y, and Square Size.

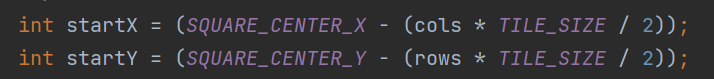


**Next piece:**

If the game is not over and there is a next piece available, the code proceeds to draw the actual preview of the next piece inside the preview area. The type of the next piece is obtained from the Tetris instance using the tetris.get Next piece type () method. The dimensions and position of the preview are calculated based on the type of the piece. The dimension variable represents the size of the piece, and the start X and start Y variables define the top left corner of the piece. The top and left variables provide the insets for the preview.

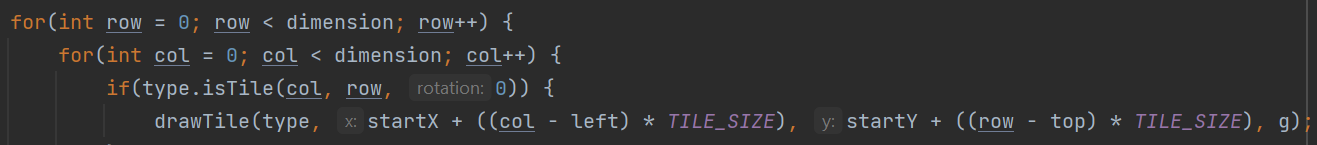
A picture containing text, font, screenshot, line

Description automatically generated



**Nested Loop:**

A nested loop is used to iterate over each tile of the piece. If a tile is present at the current position, the draw tile method is called to draw the tile on the panel. The draw tile method takes the type of the tile, its position, and the Graphics object as parameters.



**Draw Tile:**

The draw tile method is a helper method responsible for drawing a single tile on the panel. It fills the entire tile with the base color, adds shading to the bottom and right edges using the dark shading color, and adds light shading to the top and left edges using lines. The shading is achieved by drawing lines of different colors along the edges of the tile.



**Conclusion:**

The side panel class provides a visual representation of game information, controls, and the next piece preview in the Tetris game. It extends J-panel to serve as a container for holding other components. The paint component method is overridden to perform custom painting, utilizing the Graphics object to draw various elements on the panel. The code establishes a consistent visual style by using predefined constants for dimensions, positions, fonts, and colors. By referencing the Tetris instance, the Side panel ensures that the displayed information is synchronized with the game state. The draw Tile method is a helper method for drawing individual tiles on the panel with appropriate shading. Overall, the Side Panel class contributes to the overall user interface and enhances the gaming experience by providing essential information and controls.

***TileTpye.java:***

**Tile Type:**

TileTpye.java represents an Enum called tile type that defines the properties and behavior of various pieces used in a game. The game appears to be a variation of Tetris, where players manipulate falling pieces to complete rows and earn points.

A screen shot of a computer

Description automatically generated with low confidence

**Type:**

The tile type enum defines different types of tetramines, such as Type I, Type J, Type L, and so on. Each type represents a specific shape and color scheme. The enum contains properties and methods to handle the characteristics of each piece.





**Base color:**

The properties of a Tile type include the base color, light shading color, and dark shading color of the tiles for that specific piece type. These colors are represented using the java.awt.Color class. Additionally, the spawn Col and spawn Row properties indicate the initial column and row where the piece spawns on the game board.





**Dimension:**

The dimension property represents the size of the square array that holds the tiles for a piece. It determines the shape's dimensions and is usually a fixed value for each type of tetromino. The rows and cols properties specify the number of rows and columns in the piece, but their values are only valid when the rotation is 0 or 2. These properties are primarily used for displaying the next piece preview, which uses rotation 0.



**Boolean:**

The tiles property is a 2D boolean array that represents the configuration of the tiles for a specific piece type. Each element in the array corresponds to a tile position, and its boolean value indicates whether a tile exists at that position. This array stores the pattern for each rotation of the piece, allowing for dynamic shape transformations during gameplay.



**Utility Methods:**

The Tile Type Enum provides several utility methods to retrieve information about the pieces. The get Base Color (), get Light Color (), and get Dark Color () methods return the base color, light shading color, and dark shading color of a piece, respectively. These colors are used for rendering the tetrominoes on the game board.





**Get Dimension:**

The get Dimension () method returns the dimension of a piece, representing the size of the square array that holds the tiles. This method is useful for determining the dimensions of the currently active piece.



**Get spawn Colum and Row:**

The get Spawn Column () and get Spawn Row () methods return the initial column and row where a piece spawns on the game board. These methods provide the starting coordinates for each type of tetramine.

A screenshot of a computer

Description automatically generated with low confidence

**Get Rows and Cols:**

The get Rows () and get Cols () methods return the number of rows and columns in a piece. However, these values are only valid when the rotation is 0 or 2. They are typically used for rendering the next piece preview, which is displayed with rotation 0.

A picture containing text, font, screenshot

Description automatically generated

**Is Tile:**

The is Tile (int x, int y, int rotation) method checks whether a tile exists at the specified coordinates (x, y) for a given rotation of a piece. It uses the tiles array to determine the presence or absence of a tile at the given position. This method is helpful for collision detection and determining if a certain position within the game board is occupied by a tile.



**Get Left Inset:**

The get Left Inset (int rotation), get Right Inset (int rotation), get Top Inset (int rotation), and get Bottom Inset (int rotation) methods calculate the number of empty columns on the left and right sides, as well as the number of empty rows on the top and bottom sides of a piece for a given rotation. These methods are used to accurately position the tetromino on the game board during gameplay.



**Conclusion:**

The Tile Type Enum provides a comprehensive representation of different tetromino types used in a Tetris-like game. It encapsulates the properties and behavior of each piece, including colors, dimensions, rotations, and spawn locations. The provided utility methods allow for easy access to the characteristics of the pieces, aiding in rendering, collision detection, and positioning on the game board.