OOP Final Project Outline: Tetris Clone

Name: Rizky Dimas Budi Ardiansyah

Class: L2CC

Student ID: 2502016430

What is tetris?

Tetris is a game where shapes known as tetrominoes slowly descend to make full horizontal lines. When these horizontal lines are full they are deleted and cause the tetrominoes above them to drop.

How will it be implemented in Java?

The important bits to implement are moving the pieces down, removing full lines, being able to rotate tetris pieces, creating the GUI and randomly generating tetris pieces so that a player doesn't always obtain the same piece.

To start it'll be randomly generating pieces

```
    //Obtaining random numbers which are then pushed into
    //2 Dimensional Array to create random shapes based on the
    //Tetrominoes found in Tetromino
public void randomizeShape() {
            Random r = new Random();
            int x = Math.abs(r.nextInt()) % 7 + 1;
            Tetromino[] values = Tetromino.values();
            setShape(values[x]);
      }
```

In the code above we use java.util.random's Random function to obtain a random number, the number is then converted in the formula to a smaller number that can be properly used to create the Tetromino.

Secondly, there is removing full/completed lines

```
private void removeFullLines() {
  int numFullLines = 0;

  for (int i = BOARD_HEIGHT - 1; i >= 0; --i) {
    boolean lineIsFull = true;

    for (int j = 0; j < BOARD_WIDTH; ++j) {
      if (getShapePosition(j, i) == Tetromino.NullShape) {
        lineIsFull = false;
        break;
      }
    }

    if (lineIsFull) {
      ++numFullLines;

      for (int k = i; k < BOARD_HEIGHT - 1; ++k) {
        for (int j = 0; j < BOARD_WIDTH; ++j) {
          board[k * BOARD_WIDTH + j] = getShapePosition(j, k + 1);
        }
      }
    }

    //if a line is successfully removed, the score number will be increased via the code below
    if (numFullLines > 0) {
      numLinesRemoved += numFullLines;
      statusBar.setText(String.valueOf(numLinesRemoved)); //rewrites the statusbar whenever score changes
      isFallingDone = true;
      curPiece.setShape(Tetromino.NullShape);
      repaint();
    }
  }
}
```

In the above code, a variable is created to store the number of full lines. The first two for loops index through the height of the board then the width to see if a horizontal line has no clear space(only has tetris pieces) and if it is full it is removed. If lineisFull stays true then we increment numFullLines for later and begin moving any tetris squares above down equal to the amount of full lines deleted. The final portion(represented by the final if) increments numLinesRemoved by a number equal to full lines cleared and the score in the GUI is updated with the new information.

The next one is tetris piece rotation

```
//Methods to rotate tetris pieces right and left respectively
//Shifts the coordinates of the Tetris piece and then returns the new coords
public Shape rotateRight() {
        if (tetrisPiece == Tetromino.SquareShape) {
                return this;
        }
                Shape result = new Shape();
                result.tetrisPiece = tetrisPiece;

                for (int i = 0; i < 4; i++) {
                        result.setX(i, -y(i));
                        result.setY(i, x(i));
                }

        return result;
}

public Shape rotateLeft() {
        if (tetrisPiece == Tetromino.SquareShape) {
                return this;
        }
                Shape result = new Shape();
                result.tetrisPiece = tetrisPiece;

                for (int i = 0; i < 4; i++) {
                        result.setX(i, y(i));
                        result.setY(i, -x(i));
                }

        return result;
}
```

In the above code are methods to rotate 90 degrees clockwise and anticlockwise respectively. To achieve this the for loop shifts the coordinates by altering the x and y values of each individual square in a shape. Then it returns the resulting shape that has had its coordinates shifted. Additionally there is an if statement to check for whether the current Tetromino is the SquareShape which returns itself as a square can't be rotated in tetris.

The next important part is what allows the program to display the tetris pieces

```
//Every tetris piece is comprised of 4 squares
//This method draws each individual square of the tetris piece to make the full shape
private void drawSquare(Graphics g, int x, int y, Tetromino shape) {
  Color color = shape.color;
  g.setColor(color);
  g.fillRect(x + 1, y + 1, squareWidth() - 2, squareHeight() - 2);
  g.setColor(color.brighter());
  g.drawLine(x, y + squareHeight() - 1, x, y);
  g.drawLine(x, y, x + squareWidth() - 1, y);
  g.setColor(color.darker());
  g.drawLine(x + 1, y + squareHeight() - 1, x + squareWidth() - 1, y + squareHeight() - 1);
  g.drawLine(x + squareWidth() - 1, y + squareHeight() - 1, x + squareWidth() - 1, y + 1);
}
//Overriding the paint method to better suit Tetris
@Override
public void paint(Graphics g) {
  super.paint(g);
  Dimension size = getSize();
  int boardTop = (int) size.getHeight() - BOARD_HEIGHT * squareHeight();

  for (int i = 0; i < BOARD_HEIGHT; i++) {
    for (int j = 0; j < BOARD_WIDTH; ++j) {
      Tetromino shape = getShapePosition(j, BOARD_HEIGHT - i - 1);

      if (shape != Tetromino.NullShape) {
        drawSquare(g, j * squareWidth(), boardTop + i * squareHeight(), shape);
      }
    }
  }
  //in tandem with the drawSqaure method, paints the Tetromino onto the top of the screen
  if (curPiece.getTetromino() != Tetromino.NullShape) {
    for (int i = 0; i < 4; ++i) {
      int x = curX + curPiece.x(i);
      int y = curY - curPiece.y(i);
      drawSquare(g, x * squareWidth(), boardTop + (BOARD_HEIGHT - y - 1) * squareHeight(), curPiece.getTetromino());
    }
  }
}
```

There are two methods involved. The first is drawSquare which is to draw the 4 individual squares that make up any given tetromino. The second is an overloaded paint method from java.awt.Graphics. The method is modified to suit painting the board and tetris pieces on screen. The last part specifically paints the Tetromino with the help of drawSquare.

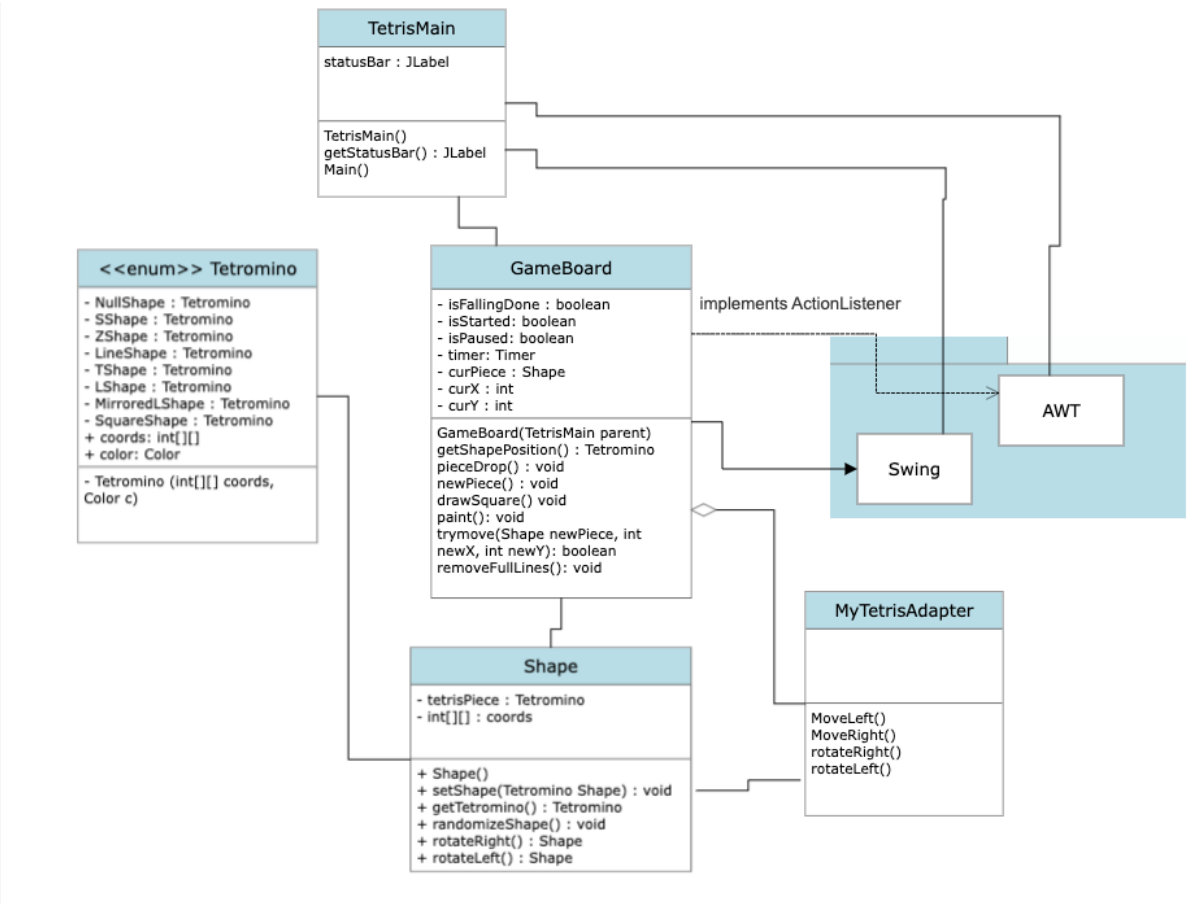
The last important implementation is the dropping of pieces

```
private void pieceDrop() {
  for (int i = 0; i < 4; i++) {
    int x = curX + curPiece.x(i);
    int y = curY - curPiece.y(i);
    board[y * BOARD_WIDTH + x] = curPiece.getTetromino();
  }
  //calls removeFullLines to check for any full lines once its done falling
  removeFullLines();

  if (!isFallingDone) {
    newPiece();
  }
}
```

The for loop in the method constantly updates where the Tetromino is located on the board until it lands on the bottom boundary or collides with another tetris piece. When it does it calls on removeFullLines to check if its landing has created a full line or not. After that it calls on newPiece to create a new piece for the player to use.

UML Class Diagram



The class diagram is complete when compared to the full program(and is missing certain components I didn't realize were needed).

What tools will be used?

Swing

Java's swing package provides the necessary tools to ease the process of making a functioning GUI. It provides a wide array of tools such as buttons, text fields, and for my project it can also allow the program to more conveniently display tetrominoes.

AWT

Much like Swing, Java's AWT package is also a package that provides the tools to make a working GUI. However unlike Swing it is more platform dependent and more heavyweight than Swing.

The tetris clone uses a mix of both to suit its needs.

References

Tetris wiki: https://en.wikipedia.org/wiki/Tetris

Java Swing tutorial: https://www.javatpoint.com/java-swing

https://zetcode.com/javagames/tetris/