Laporan Projek Akhir

Pemrograman Berorientasi Objek

Pembuatan Game Sederhana “Tetris” Dengan JavaFx

Dosen Pengampu : Didik Kurniawan S.T,. M.T,.

Di Susun Oleh:

Rahayu Indah Lestari	2317051073
Swasita Tri Widhya Anggita Cahayani	2317051018
Zahra Citra Apriliana	2317051049

Link Github :

https://github.com/zyelous/UAS-PBO.git

Fakultas Matematika dan Ilmu Pengetahuan Alam

Program Studi Ilmu Komputer

Universitas Lampung

2023/2024

Pendahuluan

• Skenario

Tetris adalah permainan berbasis puzzle yang menguji keterampilan pemain dalam menempatkan balok-balok tetromino secara strategis. Dengan mekanisme kontrol yang sederhana namun memerlukan pemikiran cepat, Tetris dirancang untuk memberikan pengalaman bermain yang seru.

Saat permainan dimulai, balok tetromino akan jatuh secara perlahan dari atas layar. Pemain dapat memutar balok menggunakan tombol rotasi dan menggeser posisi balok ke kiri atau kanan untuk menempatkannya di posisi yang diinginkan. Jika balok mencapai batas atas layar, permainan akan berakhir.

• Latar Belakang

Dalam konteks pengembangan perangkat lunak, permainan Tetris memberikan tantangan yang sangat relevan untuk menguji penerapan prinsip-prinsip Pemrograman Berorientasi Objek (PBO). Tetris memiliki elemen-elemen yang dapat direpresentasikan sebagai objek, seperti balok dengan berbagai bentuk (Line, Square, T, Z, L), ruang permainan, logika penghapusan baris, hingga penghitungan skor. Elemen-elemen ini saling berinteraksi melalui logika permainan yang kompleks namun terstruktur, sehingga sangat cocok untuk diimplementasikan dengan paradigma PBO.

Selain itu, implementasi Tetris juga melibatkan berbagai konsep penting dalam pengembangan perangkat lunak, seperti pengelolaan antarmuka pengguna grafis, animasi, dan penanganan masukan dari pengguna. Penggunaan JavaFX sebagai teknologi untuk membangun antarmuka pengguna memberikan kesempatan untuk mempraktikkan pembuatan program yang interaktif dan dinamis. Hal ini penting dalam membangun pemahaman tentang bagaimana suatu sistem perangkat lunak dapat berinteraksi dengan pengguna secara efektif.

Melalui proyek ini, diharapkan dapat dipahami lebih dalam bagaimana prinsip-prinsip PBO seperti enkapsulasi, pewarisan, polimorfisme, dan penggunaan kelas abstrak atau interface diterapkan dalam pengembangan aplikasi. Selain itu, proyek ini juga memberikan wawasan tentang pentingnya penanganan pengecualian untuk menjaga stabilitas program selama berjalan, serta penggunaan variabel atau metode statis untuk mengelola elemen-elemen global dalam permainan. Dengan membangun permainan Tetris, tidak hanya kemampuan teknis yang diasah, tetapi juga pengorganisasian logika program yang kompleks menjadi lebih modular dan mudah dipelihara.

Oleh karena itu, implementasi Tetris sebagai proyek pemrograman tidak hanya menjadi latihan teknis, tetapi juga menjadi langkah strategis dalam memahami bagaimana paradigma PBO diterapkan pada kasus nyata. Proyek ini juga dapat menjadi referensi berharga dalam membangun aplikasi permainan lain di masa depan.

• **Tujuan**

Tujuan dari proyek pengembangan permainan Tetris ini adalah:

1. Menerapkan prinsip-prinsip PBO, seperti enkapsulasi, pewarisan, polimorfisme, dan penggunaan kelas abstrak atau interface, dalam pengembangan perangkat lunak.
2. Memahami cara kerja JavaFX sebagai teknologi untuk membangun antarmuka pengguna grafis.
3. Mengasah kemampuan dalam menangani logika permainan yang melibatkan animasi, masukan pengguna, dan pengelolaan skor.
4. Mempraktikkan penanganan pengecualian untuk menjaga stabilitas program selama eksekusi.
5. Menghasilkan aplikasi yang modular, terstruktur, dan dapat dengan mudah dikembangkan atau dipelihara di masa depan.

Implementasi

- **Kode Program**

1. Kelas GameObject

```
package application;

import javafx.scene.shape.Rectangle;

public abstract class GameObject {
    protected Rectangle[] blocks = new Rectangle[4]; // Array blok
    protected int size; // Ukuran tiap blok

    public GameObject(int size) {
        this.size = size;
    }

    // Getter untuk mendapatkan ukuran blok
    public int getSize() {
        return size;
    }

    public Rectangle[] getBlocks() {
        return blocks;
    }

    public void moveLeft() {
        for (Rectangle block : blocks) {
            block.setX(block.getX() - size);
        }
    }

    public void moveRight() {
        for (Rectangle block : blocks) {
            block.setX(block.getX() + size);
        }
    }

    public void moveDown() {
        for (Rectangle block : blocks) {
            block.setY(block.getY() + size);
        }
    }

    public abstract void rotate(boolean[][] mesh);

    public void lock(boolean[][] mesh) {
        for (Rectangle block : blocks) {
            int x = (int) (block.getX() / size);
            int y = (int) (block.getY() / size);
            mesh[x][y] = true;
        }
    }
}
```

Penjelasan Kelas GameObject

- Kelas abstrak yang menjadi dasar untuk objek-objek permainan dalam aplikasi.
- Kelas ini memiliki atribut dan metode yang dapat digunakan oleh kelas turunannya untuk memanipulasi objek permainan.

Atribut

1. **Rectangle[ ] blocks**
   - Array dari objek Rectangle yang merepresentasikan blok-blok yang membentuk sebuah objek permainan.
   - Setiap Rectangle memiliki koordinat (x, y) yang dapat dimanipulasi.
2. **int size**
   - Ukuran dari setiap blok (Rectangle) dalam piksel.
   - Digunakan untuk menghitung posisi relatif dan pergerakan objek.

Konstruktor

1. **GameObject(int size)**
   - Konstruktor yang menginisialisasi atribut size untuk menentukan ukuran tiap blok.
   - Digunakan oleh kelas turunan saat membuat objek permainan.

Metode

1. **getSize()**
   - Mengembalikan nilai atribut size.
   - Berguna untuk mendapatkan informasi ukuran blok.
2. **getBlocks()**
   - Mengembalikan array blocks.
   - Berguna untuk mendapatkan akses langsung ke blok-blok penyusun objek.
3. **moveLeft()**
   - Memindahkan seluruh blok ke arah kiri sebanyak nilai size.
   - Mengurangi nilai x dari setiap Rectangle di array blocks.
4. **moveRight()**
   - Memindahkan seluruh blok ke arah kanan sebanyak nilai size.
   - Menambah nilai x dari setiap Rectangle di array blocks.
5. **moveDown()**
   - Memindahkan seluruh blok ke arah bawah sebanyak nilai size.
   - Menambah nilai y dari setiap Rectangle di array blocks.

6. **rotate(boolean[][] mesh)** ***(Abstrak)***
   - Metode abstrak yang harus diimplementasikan oleh kelas turunan untuk menangani rotasi objek.
   - Parameter mesh adalah matriks boolean yang merepresentasikan posisi objek di dalam permainan.

7. **lock(boolean[][] mesh)**
   - Mengunci posisi blok-blok pada matriks permainan (mesh).
   - Mengatur nilai mesh menjadi true sesuai koordinat tiap blok.

Penerapan

Kelas GameObject ini dirancang sebagai kerangka dasar untuk objek permainan seperti balok dalam game Tetris. Metode seperti `moveLeft`, `moveRight`, dan `moveDown` memungkinkan manipulasi posisi, sementara metode `rotate` memberikan fleksibilitas bagi kelas turunan untuk mengimplementasikan logika rotasi spesifik.

2. Kelas Form

```
package application;

import javafx.animation.RotateTransition;
import javafx.animation.TranslateTransition;
import javafx.scene.Node;
import javafx.scene.paint.Color;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Rectangle;
import javafx.util.Duration;

public class Form extends GameObject {
    private int rotationState = 0; // Keadaan rotasi saat ini

    // Constructor yang menerima size, type, dan gradient
    public Form(int size, String type) {
        super(size);  // Panggil konstruktor induk (GameObject)

        // Tentukan gradasi warna untuk setiap bentuk
        LinearGradient gradient;
        switch (type) {
            case "Line":
                gradient = new LinearGradient(0, 0, 1, 1, true, null,
                        new Stop(0, Color.CYAN), new Stop(1, Color.BLUE));
                createLine(gradient);
                break;
            case "Square":
                gradient = new LinearGradient(0, 0, 1, 1, true, null,
                        new Stop(0, Color.YELLOW), new Stop(1, Color.GOLD));
                createSquare(gradient);
                break;
            case "T":
                gradient = new LinearGradient(0, 0, 1, 1, true, null,
                        new Stop(0, Color.PURPLE), new Stop(1, Color.VIOLET));
                createT(gradient);
```

```
                break;
            case "Z":
                gradient = new LinearGradient(0, 0, 1, 1, true, null,
                        new Stop(0, Color.RED), new Stop(1, Color.DARKRED));
                createZ(gradient);
                break;
            case "L":
                gradient = new LinearGradient(0, 0, 1, 1, true, null,
                        new Stop(0, Color.ORANGE), new Stop(1,
Color.DARKORANGE));
                createL(gradient);
                break;
            default:
                throw new IllegalArgumentException("Invalid shape type.");
        }
    }

    // Method untuk membuat bentuk Line
    private void createLine(LinearGradient gradient) {
        blocks[0] = new Rectangle(size - 1, size - 1, gradient);
        blocks[1] = new Rectangle(size - 1, size - 1, gradient);
        blocks[2] = new Rectangle(size - 1, size - 1, gradient);
        blocks[3] = new Rectangle(size - 1, size - 1, gradient);
        blocks[0].setX(size * 4);
        blocks[1].setX(size * 5);
        blocks[2].setX(size * 6);
        blocks[3].setX(size * 7);
    }

    // Method untuk membuat bentuk Square
    private void createSquare(LinearGradient gradient) {
        blocks[0] = new Rectangle(size - 1, size - 1, gradient);
        blocks[1] = new Rectangle(size - 1, size - 1, gradient);
        blocks[2] = new Rectangle(size - 1, size - 1, gradient);
        blocks[3] = new Rectangle(size - 1, size - 1, gradient);
        blocks[0].setX(size * 4);
        blocks[1].setX(size * 5);
        blocks[2].setX(size * 4);
        blocks[2].setY(size);
        blocks[3].setX(size * 5);
        blocks[3].setY(size);
    }

    // Method untuk membuat bentuk T
    private void createT(LinearGradient gradient) {
        blocks[0] = new Rectangle(size - 1, size - 1, gradient);
        blocks[1] = new Rectangle(size - 1, size - 1, gradient);
        blocks[2] = new Rectangle(size - 1, size - 1, gradient);
        blocks[3] = new Rectangle(size - 1, size - 1, gradient);
        blocks[0].setX(size * 4);
        blocks[1].setX(size * 5);
        blocks[2].setX(size * 6);
        blocks[3].setX(size * 5);
        blocks[3].setY(size);
    }

    // Method untuk membuat bentuk Z
    private void createZ(LinearGradient gradient) {
        blocks[0] = new Rectangle(size - 1, size - 1, gradient);
        blocks[1] = new Rectangle(size - 1, size - 1, gradient);
        blocks[2] = new Rectangle(size - 1, size - 1, gradient);
```

```
        blocks[3] = new Rectangle(size - 1, size - 1, gradient);
        blocks[0].setX(size * 4);
        blocks[1].setX(size * 5);
        blocks[2].setX(size * 5);
        blocks[2].setY(size);
        blocks[3].setX(size * 6);
        blocks[3].setY(size);
    }

    // Method untuk membuat bentuk L
    private void createL(LinearGradient gradient) {
        blocks[0] = new Rectangle(size - 1, size - 1, gradient);
        blocks[1] = new Rectangle(size - 1, size - 1, gradient);
        blocks[2] = new Rectangle(size - 1, size - 1, gradient);
        blocks[3] = new Rectangle(size - 1, size - 1, gradient);
        blocks[0].setX(size * 4);
        blocks[1].setX(size * 5);
        blocks[2].setX(size * 6);
        blocks[3].setX(size * 6);
        blocks[3].setY(size);
    }

    // Animasi rotasi
    private void animateRotation(Node block) {
        RotateTransition rotate = new RotateTransition(Duration.millis(200),
block);
        rotate.setByAngle(90); // Putar 90 derajat
        rotate.setCycleCount(1);
        rotate.play();
    }

    @Override
    public void rotate(boolean[][] mesh) {
        if (blocks.length <= 1) return; // Square tidak perlu rotasi

        // Menyimpan posisi sebelum rotasi untuk membatalkan jika keluar dari
grid
        double[] originalPositionsX = new double[blocks.length];
        double[] originalPositionsY = new double[blocks.length];

        for (int i = 0; i < blocks.length; i++) {
            originalPositionsX[i] = blocks[i].getX();
            originalPositionsY[i] = blocks[i].getY();
        }

        double centerX = blocks[1].getX();  // Pusat rotasi
        double centerY = blocks[1].getY();  // Pusat rotasi

        // Rotasi blok
        for (Rectangle block : blocks) {
            double relativeX = block.getX() - centerX;
            double relativeY = block.getY() - centerY;

            block.setX(centerX - relativeY);
            block.setY(centerY + relativeX);

            animateRotation(block); // Animasi rotasi
        }

        // Memastikan rotasi tidak keluar dari grid
        if (!canMove(this, mesh)) {
```

```
                // Batalkan rotasi jika keluar dari grid
                for (int i = 0; i < blocks.length; i++) {
                    blocks[i].setX(originalPositionsX[i]);
                    blocks[i].setY(originalPositionsY[i]);
                }
            }
        }

        // Mengganti WIDTH dan HEIGHT dengan Tetris.WIDTH dan Tetris.HEIGHT
        public boolean canMove(Form form, boolean[][] mesh) {
            for (Rectangle block : form.getBlocks()) {
                int x = (int) (block.getX() / size);
                int y = (int) (block.getY() / size);

                // Periksa apakah blok keluar dari grid atau bertabrakan dengan
blok lain
                if (x < 0 || x >= Tetris.WIDTH || y >= Tetris.HEIGHT ||
mesh[x][y]) {
                    return false;
                }
            }
            return true;
        }
    }
```

Penjelasan Kelas Form

- Kelas turunan dari GameObject yang merepresentasikan berbagai bentuk geometris dalam permainan (seperti Line, Square, T, Z, dan L).
- Menyediakan logika untuk membuat bentuk-bentuk ini dengan warna gradasi, memutar bentuk, dan memeriksa validitas gerakan dalam grid permainan.

Atribut

1. **int rotationState**
    - Menyimpan keadaan rotasi saat ini untuk bentuk yang dapat diputar.
    - Nilainya diatur dan digunakan untuk mengontrol logika rotasi.

Konstruktor

1. **Form(int size, String type)**
    - Menerima parameter size (ukuran tiap blok) dan type (jenis bentuk).
    - Memanggil konstruktor kelas induk untuk mengatur ukuran blok.
    - Berdasarkan type, memanggil metode spesifik untuk membuat bentuk (contoh: `createLine` untuk bentuk Line).
    - Mengatur gradasi warna menggunakan `LinearGradient`.

Metode untuk Membuat Bentuk

1. **createLine(LinearGradient gradient)**
    - Membuat bentuk garis horizontal yang terdiri dari 4 blok.
    - Setiap blok diatur posisinya secara berurutan pada koordinat x yang berbeda.

2. **createSquare(LinearGradient gradient)**
    - Membuat bentuk persegi 2x2 menggunakan 4 blok.
    - Mengatur posisi blok untuk membentuk pola kotak.
3. **createT(LinearGradient gradient)**
    - Membuat bentuk seperti huruf T dengan 4 blok.
    - Posisi blok diatur untuk membentuk pola T.
4. **createZ(LinearGradient gradient)**
    - Membuat bentuk seperti huruf Z dengan 4 blok.
    - Blok diatur dalam posisi zig-zag.
5. **createL(LinearGradient gradient)**
    - Membuat bentuk seperti huruf L dengan 4 blok.
    - Posisi blok diatur untuk membentuk pola L.

Metode Lain

1. **animateRotation(Node block)**
    - Menambahkan animasi rotasi pada sebuah blok (Node) menggunakan `RotateTransition.`
    - Memutar blok sebesar 90 derajat dalam durasi 200 ms.
2. **rotate(boolean[][] mesh)**
    - Mengimplementasikan rotasi untuk bentuk yang dapat diputar (kecuali Square).
    - Menghitung posisi baru tiap blok berdasarkan pusat rotasi.
    - Memastikan rotasi valid (tidak keluar grid atau bertabrakan), dan membatalkan jika tidak valid.
    - Menambahkan animasi rotasi pada setiap blok.
3. **canMove(Form form, boolean[][] mesh)**
    - Memeriksa apakah bentuk dapat bergerak atau berputar dalam grid.
    - Memastikan posisi blok tidak keluar dari batas grid (Tetris.WIDTH dan Tetris.HEIGHT) atau bertabrakan dengan blok lain.

Catatan Khusus

1. **Gradasi Warna**
    - Gradasi warna (`LinearGradient`) digunakan untuk memberikan estetika pada setiap bentuk, dengan kombinasi warna berbeda untuk setiap jenis.

2. **Integrasi dengan Grid Permainan**

   - mesh adalah matriks boolean yang merepresentasikan grid permainan. Metode seperti `rotate` dan `canMove` memanfaatkan matriks ini untuk validasi posisi bentuk.

3. **Animasi**

   - Rotasi bentuk diperkaya dengan animasi menggunakan `RotateTransition`, memberikan efek visual yang menarik selama permainan.

3. Kelas Controller

```
package application;

import javafx.scene.shape.Rectangle;

public class Controller {

    // Memeriksa apakah form bisa bergerak ke kiri
    public boolean canMoveLeft(GameObject form, boolean[][] mesh) {
        // Jika game over, tidak ada pergerakan
        if (Tetris.gameOver) return false;

        // Memeriksa apakah blok bisa bergerak ke kiri tanpa keluar dari grid
atau bertabrakan
        for (Rectangle block : form.getBlocks()) {
            int x = (int) (block.getX() / form.getSize());
            int y = (int) (block.getY() / form.getSize());

            // Periksa apakah blok berada di batas kiri atau bertabrakan
dengan blok lain
            if (x - 1 < 0 || mesh[x - 1][y]) {
                return false;
            }
        }
        return true;
    }

    // Memeriksa apakah form bisa bergerak ke kanan
    public boolean canMoveRight(GameObject form, boolean[][] mesh) {
        // Jika game over, tidak ada pergerakan
        if (Tetris.gameOver) return false;

        // Memeriksa apakah blok bisa bergerak ke kanan tanpa keluar dari grid
atau bertabrakan
        for (Rectangle block : form.getBlocks()) {
            int x = (int) (block.getX() / form.getSize());
            int y = (int) (block.getY() / form.getSize());

            // Periksa apakah blok berada di batas kanan atau bertabrakan
dengan blok lain
            if (x + 1 >= Tetris.WIDTH || mesh[x + 1][y]) {
                return false;
            }
        }
        return true;
    }
```

```
    // Memeriksa apakah form bisa bergerak ke bawah
    public boolean canMoveDown(GameObject form, boolean[][] mesh) {
        // Jika game over, tidak ada pergerakan
        if (Tetris.gameOver) return false;

        // Memeriksa apakah blok bisa bergerak ke bawah tanpa keluar dari grid
atau bertabrakan
        for (Rectangle block : form.getBlocks()) {
            int x = (int) (block.getX() / form.getSize());
            int y = (int) (block.getY() / form.getSize());

            // Periksa apakah blok berada di batas bawah atau bertabrakan
dengan blok lain
            if (y + 1 >= Tetris.HEIGHT || mesh[x][y + 1]) {
                return false;
            }
        }
        return true;
    }
}
```

Penjelasan Kelas Controller

- Kelas ini bertanggung jawab untuk mengontrol pergerakan bentuk (GameObject) dalam grid permainan.
- Memastikan bentuk tidak keluar dari grid atau bertabrakan dengan blok lain saat bergerak ke kiri, kanan, atau bawah.

Metode

1. **canMoveLeft(GameObject form, boolean[][] mesh)**
    - **Deskripsi:** Memeriksa apakah bentuk dapat bergerak ke kiri.
    - **Logika:**
        - Iterasi melalui semua blok dalam bentuk (form).
        - Hitung koordinat x dan y blok berdasarkan posisinya dalam grid.
        - Periksa apakah:
            - Koordinat x setelah pergerakan ke kiri menjadi kurang dari 0 (batas kiri grid).
            - Ada blok lain di posisi tersebut dalam matriks mesh.
        - Jika salah satu kondisi di atas terpenuhi, bentuk tidak dapat bergerak ke kiri.
    - **Pengembalian:**
        - true jika pergerakan ke kiri memungkinkan.
        - false jika tidak memungkinkan.

2. **canMoveRight(GameObject form, boolean[][] mesh)**
    - **Deskripsi:** Memeriksa apakah bentuk dapat bergerak ke kanan.
    - **Logika:**
        - Iterasi melalui semua blok dalam bentuk.
        - Hitung koordinat x dan y blok.
        - Periksa apakah:
            - Koordinat x setelah pergerakan ke kanan melebihi batas kanan grid (Tetris.WIDTH).
            - Ada blok lain di posisi tersebut dalam mesh.
        - Jika salah satu kondisi di atas terpenuhi, bentuk tidak dapat bergerak ke kanan.
    - **Pengembalian:**
        - true jika pergerakan ke kanan memungkinkan.
        - false jika tidak memungkinkan.
3. **canMoveDown(GameObject form, boolean[][] mesh)**
    - **Deskripsi:** Memeriksa apakah bentuk dapat bergerak ke bawah.
    - **Logika:**
        - Iterasi melalui semua blok dalam bentuk.
        - Hitung koordinat x dan y blok.
        - Periksa apakah:
            - Koordinat y setelah pergerakan ke bawah melebihi batas bawah grid (Tetris.HEIGHT).
            - Ada blok lain di posisi tersebut dalam mesh.
        - Jika salah satu kondisi di atas terpenuhi, bentuk tidak dapat bergerak ke bawah.
    - **Pengembalian:**
        - true jika pergerakan ke bawah memungkinkan.
        - false jika tidak memungkinkan.

Catatan Khusus

1. **Parameter mesh**
    - Matriks boolean yang merepresentasikan grid permainan. true berarti posisi tersebut sudah ditempati oleh blok, sedangkan false berarti kosong.
2. **Kondisi Game Over**
    - Setiap metode memeriksa apakah permainan telah berakhir (Tetris.gameOver). Jika ya, pergerakan bentuk tidak diperbolehkan.
3. **Batas Grid**
    - Tetris.WIDTH dan Tetris.HEIGHT digunakan untuk memvalidasi apakah bentuk tetap berada dalam batas grid.

Fungsi Kelas Controller

Kelas Controller memastikan integritas pergerakan bentuk dalam permainan. Dengan memeriksa setiap potensi gerakan, kelas ini mencegah kesalahan seperti keluar dari grid atau bertabrakan dengan blok lain, menjaga kelancaran logika permainan.

4. Kelas Tetris

```
package application;

import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;
import javafx.scene.shape.Line;
import javafx.scene.shape.Rectangle;

import java.util.Random;

public class Tetris extends Application {

    public static final int SIZE = 25;
    public static final int WIDTH = 10;
    public static final int HEIGHT = 20;
    private final Pane gamePane = new Pane();
    private final boolean[][] MESH = new boolean[WIDTH][HEIGHT];
    private GameObject currentForm;
    private final Controller controller = new Controller();
    private static int score = 0;
```

```
    private static int lines = 0;
    private final Text scoreText = new Text("Score: 0");
    private final Text linesText = new Text("Lines: 0");

    private Timeline timeline;
    public static boolean gameOver = false;

    @Override
    public void start(Stage stage) {
        gamePane.setPrefSize(WIDTH * SIZE, HEIGHT * SIZE);
        gamePane.setStyle("-fx-background-color: #000000;"); // Warna latar
belakang hitam
        drawGrid();

        // Tampilan awal
        VBox startMenu = new VBox(10);
        startMenu.setStyle("-fx-alignment: center;");

        Text titleText = new Text("TETRIS");
        titleText.setFont(Font.font("Verdana", 40));
        titleText.setFill(Color.RED);

        Button startButton = new Button("Start");
        startButton.setStyle("-fx-font-size: 16px;");
        startButton.setOnAction(e -> {
            stage.setScene(createGameScene(stage));
            spawnNewForm();
            timeline.play();
        });

        startMenu.getChildren().addAll(titleText, startButton);
        Scene startScene = new Scene(new StackPane(startMenu), WIDTH * SIZE +
100, HEIGHT * SIZE);
        stage.setScene(startScene);
        stage.setTitle("Tetris");
        stage.show();
    }

    private Scene createGameScene(Stage stage) {
        scoreText.setX(WIDTH * SIZE + 10);
        scoreText.setY(20);
        linesText.setX(WIDTH * SIZE + 10);
        linesText.setY(50);

        Pane root = new Pane(gamePane, scoreText, linesText);
        Scene scene = new Scene(root, WIDTH * SIZE + 100, HEIGHT * SIZE);

        scene.setOnKeyPressed(event -> {
            if (gameOver) return;

            try {
                switch (event.getCode()) {
                    case LEFT:
                        if (controller.canMoveLeft(currentForm, MESH))
currentForm.moveLeft();
                        break;
                    case RIGHT:
                        if (controller.canMoveRight(currentForm, MESH))
currentForm.moveRight();
                        break;
                    case DOWN:
```

```
                        if (controller.canMoveDown(currentForm, MESH))
currentForm.moveDown();
                        else lockAndSpawnNewForm();
                        break;
                    case UP:
                        currentForm.rotate(MESH);
                        break;
                    default:
                        break;
                }
            } catch (Exception ex) {
                showError("Error processing input: " + ex.getMessage());
            }
        });

        timeline = new Timeline(new KeyFrame(Duration.millis(500), e -> {
            if (!gameOver) {
                try {
                    if (controller.canMoveDown(currentForm, MESH)) {
                        currentForm.moveDown();
                    } else {
                        lockAndSpawnNewForm();
                        if (isGameOver()) {
                            gameOver = true;
                            timeline.stop();
                            stage.setScene(createGameOverScene(stage));
                        }
                    }
                } catch (Exception ex) {
                    showError("Error during game update: " + ex.getMessage());
                }
            }
        }));
        timeline.setCycleCount(Timeline.INDEFINITE);

        return scene;
    }

    private Scene createGameOverScene(Stage stage) {
        VBox gameOverMenu = new VBox(10);
        gameOverMenu.setStyle("-fx-alignment: center;");

        Text gameOverText = new Text("GAME OVER");
        gameOverText.setFont(Font.font("Verdana", 30));
        gameOverText.setFill(Color.RED);

        Button tryAgainButton = new Button("Try Again");
        tryAgainButton.setStyle("-fx-font-size: 16px;");
        tryAgainButton.setOnAction(e -> {
            resetGame();
            stage.setScene(createGameScene(stage));
            spawnNewForm();
            timeline.play();
        });

        gameOverMenu.getChildren().addAll(gameOverText, tryAgainButton);
        return new Scene(new StackPane(gameOverMenu), WIDTH * SIZE + 100,
HEIGHT * SIZE);
    }

    private void resetGame() {
```

```
    gameOver = false;
    score = 0;
    lines = 0;
    updateScoreAndLines();
    gamePane.getChildren().clear();
    for (int x = 0; x < WIDTH; x++) {
        for (int y = 0; y < HEIGHT; y++) {
            MESH[x][y] = false;
        }
    }
    drawGrid();
}

private void drawGrid() {
    for (int i = 0; i <= WIDTH; i++) {
        Line line = new Line(i * SIZE, 0, i * SIZE, HEIGHT * SIZE);
        line.setStroke(Color.GRAY);
        line.setStrokeWidth(1);
        gamePane.getChildren().add(line);
    }

    for (int i = 0; i <= HEIGHT; i++) {
        Line line = new Line(0, i * SIZE, WIDTH * SIZE, i * SIZE);
        line.setStroke(Color.GRAY);
        line.setStrokeWidth(1);
        gamePane.getChildren().add(line);
    }
}

private void spawnNewForm() {
    if (gameOver) return;

    try {
        Random random = new Random();
        int shapeType = random.nextInt(5);

        // Menggunakan polimorfisme di sini
        switch (shapeType) {
            case 0: currentForm = new Form(SIZE, "Line"); break;
            case 1: currentForm = new Form(SIZE, "Square"); break;
            case 2: currentForm = new Form(SIZE, "T"); break;
            case 3: currentForm = new Form(SIZE, "Z"); break;
            case 4: currentForm = new Form(SIZE, "L"); break;
        }

        gamePane.getChildren().addAll(currentForm.getBlocks());
    } catch (Exception e) {
        showError("Error spawning new form: " + e.getMessage());
    }
}

private void lockAndSpawnNewForm() {
    try {
        currentForm.lock(MESH);
        clearLines();
        spawnNewForm();
    } catch (Exception e) {
        showError("Error saat mengunci form: " + e.getMessage());
    }
}
```

```
    private void clearLines() {
        for (int y = 0; y < HEIGHT; y++) {
            boolean fullLine = true;

            for (int x = 0; x < WIDTH; x++) {
                if (!MESH[x][y]) {
                    fullLine = false;
                    break;
                }
            }

            if (fullLine) {
                for (int x = 0; x < WIDTH; x++) {
                    MESH[x][y] = false;
                    int finalX = x;
                    int finalY = y;

                    gamePane.getChildren().removeIf(node -> {
                        if (node instanceof Rectangle) {
                            Rectangle rect = (Rectangle) node;
                            return rect.getX() == finalX * SIZE && rect.getY()
== finalY * SIZE;
                        }
                        return false;
                    });
                }

                for (int i = y - 1; i >= 0; i--) {
                    for (int x = 0; x < WIDTH; x++) {
                        if (MESH[x][i]) {
                            MESH[x][i] = false;
                            MESH[x][i + 1] = true;

                            final int finalX = x;
                            final int finalY = i;

                            gamePane.getChildren().forEach(node -> {
                                if (node instanceof Rectangle) {
                                    Rectangle rect = (Rectangle) node;
                                    if (rect.getX() == finalX * SIZE &&
rect.getY() == finalY * SIZE) {
                                        rect.setY(rect.getY() + SIZE);
                                    }
                                }
                            });
                        }
                    }
                }

                score += 10;
                lines++;
                updateScoreAndLines();
            }
        }
    }

    private void updateScoreAndLines() {
        scoreText.setText("Score: " + score);
        linesText.setText("Lines: " + lines);
    }
```

```
    private boolean isGameOver() {
        for (int x = 0; x < WIDTH; x++) {
            if (MESH[x][0]) {
                return true;
            }
        }
        return false;
    }

    private void showError(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    public static int getScore() {
        return score;
    }

    public static void setScore(int score) {
        Tetris.score = score;
    }

    public static int getLines() {
        return lines;
    }

    public static void setLines(int lines) {
        Tetris.lines = lines;
    }

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

Penjelasan Kelas Tetris

- Kelas ini adalah turunan dari Application yang digunakan untuk membuat dan menjalankan aplikasi JavaFX.
- Kelas ini menangani logika utama dari permainan Tetris, termasuk tampilan, interaksi pengguna, pembaruan skor, serta logika permainan seperti pergerakan dan pengecekan kondisi game over.

Deklarasi Variabel

1. **SIZE**: Ukuran dari setiap kotak dalam grid, yaitu 25 piksel.
2. **WIDTH dan HEIGHT**: Lebar dan tinggi grid permainan dalam satuan kotak. Grid ini berukuran 10x20.
3. **gamePane**: Pane yang digunakan untuk menggambar dan menampilkan elemen-elemen permainan seperti blok dan garis.

4. **MESH**: Matriks boolean yang menyimpan informasi mengenai status setiap sel dalam grid (terisi atau kosong).
5. **currentForm**: Objek yang mewakili bentuk Tetris yang sedang dimainkan.
6. **controller**: Objek dari kelas Controller yang mengontrol pergerakan dan logika permainan.
7. **score dan lines**: Variabel statis yang menyimpan skor dan jumlah garis yang telah terhapus.
8. **scoreText dan linesText**: Objek Text yang digunakan untuk menampilkan skor dan jumlah garis yang telah dihapus di sisi kanan layar.
9. **timeline**: Objek Timeline yang mengatur pembaruan permainan setiap interval waktu.
10. **gameOver**: Variabel statis yang menunjukkan apakah permainan telah berakhir atau belum.

Metode Utama

1. **start(Stage stage)**:
    - Metode ini adalah titik masuk aplikasi JavaFX. Di sini, tampilan awal permainan (menu mulai) dibuat, dan ketika tombol "Start" ditekan, permainan dimulai.
2. **createGameScene(Stage stage)**:
    - Metode ini membuat scene untuk permainan utama, dengan menambahkan event handler untuk tombol arah (panah) yang digunakan untuk menggerakkan bentuk Tetris.
    - Menggunakan Timeline untuk memperbarui posisi bentuk setiap 500 milidetik.
    - Jika bentuk tidak bisa bergerak ke bawah, bentuk "dihapus" dan bentuk baru dihasilkan.
3. **createGameOverScene(Stage stage)**:
    - Metode ini menampilkan tampilan game over dengan opsi untuk memulai ulang permainan.
4. **resetGame()**:
    - Mengatur ulang permainan (skor, garis, dan grid), serta menggambar ulang grid.
5. **drawGrid()**:
    - Menggambar grid permainan, dengan garis-garis vertikal dan horizontal menggunakan objek Line.

6. **spawnNewForm()**:
    - Membuat bentuk baru secara acak (menggunakan polimorfisme) dan menambahkannya ke game pane.

7. **lockAndSpawnNewForm()**:
    - Mengunci posisi bentuk yang sudah mencapai bagian bawah grid dan mengecek apakah ada garis penuh yang harus dihapus.
    - Setelah itu, bentuk baru dihasilkan.
8. **clearLines()**:
    - Menghapus garis yang terisi penuh dan memperbarui posisi blok di atasnya.
9. **updateScoreAndLines()**:
    - Memperbarui teks yang menampilkan skor dan jumlah garis yang telah dihapus.
10. **isGameOver()**:
    - Mengecek apakah permainan telah berakhir. Jika ada blok yang menempati baris paling atas grid, permainan berakhir.
11. **showError(String message)**:
    - Menampilkan dialog error jika terjadi kesalahan selama permainan (misalnya, dalam pengolahan input atau pembaruan status permainan).

Event Handling

- Ketika tombol arah ditekan, pergerakan bentuk akan diperiksa menggunakan objek Controller untuk memastikan bentuk tidak keluar dari grid atau bertabrakan dengan blok lain.
- LEFT, RIGHT, DOWN, dan UP adalah tombol yang menggerakkan bentuk Tetris (kiri, kanan, bawah, dan rotasi).

Logika Game

- Permainan berlanjut dengan menurunkan bentuk secara otomatis setiap setengah detik.
- Setiap kali bentuk tidak bisa bergerak lebih lanjut, ia akan "terkunci" (ditambahkan ke grid tetap) dan bentuk baru akan dimunculkan.
- Jika garis terisi penuh, garis tersebut dihapus, dan pemain mendapatkan skor.

Tampilan dan Tampilan Awal

- Pada tampilan awal, ada menu dengan judul permainan dan tombol "Start". Ketika tombol ditekan, permainan dimulai dan scene permainan utama akan ditampilkan.

Kelas Tetris ini menciptakan permainan Tetris klasik dengan kontrol dasar (gerakan dan rotasi bentuk) serta sistem untuk menghitung skor dan menghapus garis. Seluruh logika permainan dijalankan dalam loop berbasis Timeline, dengan pengendalian pergerakan menggunakan objek Controller.

Pembahasan

1. Encapsulation

Encapsulation adalah konsep di mana data disembunyikan dari akses langsung oleh kelas lain dan hanya bisa diakses melalui metode yang disediakan. Ini dilakukan untuk memastikan integritas data dan mencegah perubahan yang tidak sah atau tidak terkontrol.

- **Contoh dalam kode:**
    - **Atribut dalam kelas Tetris:** Variabel seperti score, lines, dan gameOver disembunyikan (private) dan tidak dapat diakses langsung oleh kelas lain. Akses terhadap nilai-nilai ini hanya dilakukan melalui metode getter dan setter.
    - **Metode getter/setter:**
        - getScore() dan setScore(int score) digunakan untuk mengakses dan memperbarui nilai score.
        - getLines() dan setLines(int lines) digunakan untuk mengakses dan memperbarui nilai lines.
        - Penggunaan getter dan setter ini membantu mengontrol perubahan nilai atribut secara lebih terstruktur dan aman.
    - **Contoh lainnya:** Dalam kelas Controller, pergerakan bentuk dalam game dipastikan hanya dilakukan dengan pengecekan tertentu (seperti canMoveLeft(), canMoveRight(), dll.), sehingga data posisi blok tidak dapat diubah sembarangan.

2. Inheritance

Inheritance (pewarisan) adalah konsep di mana sebuah kelas dapat mewarisi atribut dan metode dari kelas lain. Konsep ini memungkinkan kode untuk digunakan kembali dan memperkenalkan hierarki kelas.

- **Contoh dalam kode:**
    - **Tetris mewarisi Application:** Kelas Tetris adalah subclass dari Application (kelas dasar dari JavaFX yang menyediakan kerangka untuk aplikasi berbasis GUI). Dengan mewarisi kelas Application, kelas Tetris dapat menggunakan metode-metode seperti launch(args) untuk memulai aplikasi.
    - **Form dan GameObject:** Di dalam game, Form adalah subclass dari GameObject yang mendefinisikan bentuk-bentuk Tetris. Form mewarisi properti dan metode dari GameObject, seperti getBlocks() yang mengembalikan blok-blok dalam bentuk. Ini memungkinkan berbagai

bentuk (seperti Line, Square, dll.) untuk berbagi logika yang sama meskipun memiliki implementasi yang berbeda.

3. Polymorphism

Polymorphism memungkinkan objek dari kelas yang berbeda untuk diperlakukan secara seragam, meskipun mungkin memiliki implementasi yang berbeda. Dalam konteks ini, objek dari kelas yang berbeda dapat diperlakukan melalui antarmuka atau kelas dasar yang sama.

- **Contoh dalam kode:**
    - **Polimorfisme pada kelas Form:** Di dalam metode spawnNewForm(), bentuk yang akan muncul dipilih secara acak dari lima jenis yang berbeda (Line, Square, T, Z, L). Meskipun setiap bentuk adalah objek yang berbeda, mereka semua diperlakukan secara seragam sebagai objek GameObject. Metode seperti getBlocks() dan moveDown() dapat dipanggil pada objek Form tanpa memedulikan bentuk spesifiknya, berkat pewarisan dari GameObject.
    - **Penggunaan Polimorfisme:** Dengan polimorfisme, kita bisa mengatur bentuk-bentuk tersebut secara umum tanpa perlu menulis kode terpisah untuk setiap bentuk, seperti pada metode spawnNewForm() yang menggunakan polimorfisme untuk membuat objek bentuk yang berbeda.

4. Abstract Class atau Interface

Abstract Class atau **Interface** adalah kontrak yang mendefinisikan metode yang harus diimplementasikan oleh kelas turunan. Di Java, kelas abstrak tidak dapat diinstansiasi langsung dan dapat mengandung metode yang belum diimplementasikan (metode abstrak).

- **Contoh dalam kode:**
    - **Penggunaan Kelas GameObject:** Kelas GameObject seharusnya bisa menjadi kelas abstrak, yang mendefinisikan kontrak dasar bagi objek-objek permainan seperti bentuk Tetris. Misalnya, getBlocks(), moveDown(), dan rotate() adalah metode yang harus diimplementasikan oleh setiap subclass GameObject (seperti Form).
    - Walaupun kelas GameObject tidak secara eksplisit ditandai sebagai kelas abstrak dalam kode ini, ide tersebut tetap relevan dalam struktur permainan ini, di mana Form adalah subclass yang mengimplementasikan metode dasar.

5. Static Modifier

Static modifier digunakan untuk mendeklarasikan variabel atau metode yang dapat diakses tanpa membuat instance dari kelas tersebut. Variabel atau metode static adalah milik kelas, bukan milik objek.

- **Contoh dalam kode:**
    - **Variabel score, lines, dan gameOver:** Variabel ini dideklarasikan sebagai static karena nilainya harus dibagikan di antara semua objek Tetris. Dengan membuatnya static, nilai ini dapat diakses atau dimodifikasi tanpa perlu membuat objek baru dari kelas Tetris. Ini memungkinkan informasi seperti skor dan garis untuk konsisten di seluruh permainan.
    - **Metode getScore(), setScore(), getLines(), setLines():** Karena score dan lines adalah variabel statis, metode getter dan setter juga bersifat statis untuk dapat mengaksesnya tanpa instance dari kelas Tetris.

6. Error/Exception Handling

Error/Exception Handling adalah cara untuk menangani masalah yang muncul selama eksekusi program, seperti kesalahan input atau kesalahan logika.

- **Contoh dalam kode:**
    - **Pengecualian pada pergerakan:** Dalam createGameScene(), terdapat blok kode yang menangani input pengguna dan pergerakan bentuk. Jika ada kesalahan saat memproses input, exception akan dilempar dan ditangani oleh catch yang memanggil metode showError(), yang menampilkan pesan kesalahan kepada pengguna.
    - **Metode showError(String message):** Metode ini menampilkan dialog alert yang memberi tahu pengguna tentang kesalahan yang terjadi, misalnya kesalahan saat memproses input atau saat pembaruan status permainan. Ini membantu menjaga stabilitas aplikasi dengan memberi tahu pengguna tentang masalah tanpa menghentikan aplikasi secara mendadak

Kesimpulan

Dalam tugas ini, penerapan konsep Pemrograman Berorientasi Objek (PBO) pada implementasi permainan Tetris telah dijelaskan dengan menggunakan berbagai prinsip dasar PBO. Berikut adalah kesimpulan mengenai penerapan PBO dalam kode program:

1. **Encapsulation** diterapkan dengan menyembunyikan atribut-atribut penting seperti score, lines, dan gameOver agar tidak dapat diakses langsung dari luar kelas. Akses terhadap atribut-atribut tersebut dilakukan melalui metode getter dan setter. Dengan cara ini, data permainan tetap terlindungi dan hanya dapat diubah melalui metode yang sudah ditentukan.
2. **Inheritance** digunakan untuk memanfaatkan pewarisan kelas, di mana kelas Tetris mewarisi kelas Application untuk membangun aplikasi berbasis JavaFX. Selain itu, kelas Form mewarisi kelas GameObject, memungkinkan bentuk-bentuk Tetris yang berbeda untuk berbagi logika dasar yang sama meskipun memiliki implementasi yang berbeda.
3. **Polymorphism** diterapkan dengan cara objek-objek dari kelas yang berbeda (seperti Line, Square, T, Z, dan L) dapat diperlakukan secara seragam. Semua objek ini adalah turunan dari kelas GameObject, yang memiliki metode-metode dasar yang sama seperti moveDown(), sehingga mempermudah pengelolaan berbagai jenis bentuk dalam permainan.
4. **Abstract Class atau Interface** digunakan melalui kelas GameObject, yang mendefinisikan kontrak dasar untuk bentuk-bentuk dalam permainan. Metode seperti getBlocks() dan moveDown() didefinisikan dalam kelas ini dan diimplementasikan oleh kelas-kelas turunan seperti Form. Hal ini memastikan bahwa setiap objek yang merupakan bentuk Tetris memiliki perilaku yang konsisten meskipun bentuknya berbeda.
5. **Static Modifier** digunakan pada variabel dan metode yang perlu diakses tanpa harus membuat instance baru dari kelas, seperti variabel score dan lines. Penggunaan static memastikan data permainan tetap konsisten dan dapat diakses secara global, tanpa perlu membuat objek baru untuk setiap perubahan status permainan.

6. **Error/Exception Handling** diterapkan untuk menangani pengecualian yang mungkin terjadi selama permainan, misalnya saat memproses input pengguna atau memperbarui status permainan. Metode showError() digunakan untuk menampilkan pesan kesalahan yang jelas kepada pengguna, yang membantu menjaga kestabilan aplikasi dan mencegah program dari crash.

Secara keseluruhan, penerapan prinsip-prinsip PBO dalam permainan Tetris ini menciptakan aplikasi yang lebih terstruktur, modular, dan fleksibel. Hal ini memungkinkan pengembangan dan pemeliharaan kode yang lebih mudah serta memberikan pengalaman pengguna yang lebih baik dalam bermain. Konsep-konsep PBO yang diterapkan mendukung pengelolaan objek-objek permainan secara efisien dan memberikan dasar yang kokoh untuk pengembangan lebih lanjut.