

LAPORAN TUGAS BESAR
PEMROGRAMAN BERORIENTASI OBJEK



DISUSUN OLEH
MUHAMAD ZIDAN DICKY NASUHA (L0124061)
MUHAMMAD IRFAN (L0124063)
MUHAMMAD RAFI RAIHAN SONJAYA (L0124065)

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA
UNIVERSITAS SEBELAS MARET

2025

BAB I

PENDAHULUAN

1.1 Latar Belakang

Minat siswa terhadap mata pelajaran sejarah Indonesia saat ini cenderung menurun. Banyak siswa menganggap sejarah sebagai mata pelajaran yang membosankan, penuh hafalan, dan kurang relevan dengan kehidupan mereka. Penyampaian materi yang masih dominan menggunakan metode konvensional seperti ceramah atau membaca buku teks membuat siswa kesulitan untuk membangun keterlibatan emosional dan imajinasi terhadap peristiwa sejarah.

Di era digital dan serba visual, siswa lebih tertarik pada media interaktif seperti game, video, dan aplikasi yang dapat memberikan pengalaman belajar yang menyenangkan. Sayangnya, penggunaan media berbasis teknologi dalam pembelajaran sejarah masih belum optimal. Akibatnya, pembelajaran sejarah kurang mampu menumbuhkan rasa ingin tahu, penghargaan terhadap nilai perjuangan bangsa, serta pemahaman yang mendalam tentang identitas nasional.

Melihat kondisi ini, diperlukan sebuah inovasi pembelajaran yang dapat menghadirkan pengalaman belajar sejarah Indonesia secara lebih menarik, immersif, dan relevan bagi generasi muda. Salah satu media yang potensial untuk mencapai tujuan tersebut adalah game edukasi yang mampu menggabungkan unsur hiburan dan edukasi sekaligus.

1.2 Solusi

Sebagai solusi, kami menciptakan sebuah program dengan fitur-fitur sebagai berikut :

1. Penyajian Video Sejarah yang Menarik dan Informatif

Aplikasi ini menyediakan berbagai video pembelajaran sejarah Indonesia yang dikemas secara kreatif dan modern. Video-video ini tidak hanya berisi narasi monoton, tetapi dilengkapi dengan:

- Animasi visual yang menggambarkan adegan sejarah secara dramatis dan mudah dipahami.
- Ilustrasi dan infografis untuk membantu menjelaskan peristiwa penting, tokoh, atau konsep.
- Penyampaian narasi yang ringan, seru, dan relevan bagi siswa, sehingga materi sejarah terasa lebih dekat dan tidak membosankan.

Dengan pendekatan ini, siswa dapat menyerap materi tanpa merasa terbebani menghafal, karena visual yang menarik membantu mereka memahami konteks dan makna peristiwa sejarah.

2. Game Interaktif yang Menghadirkan Pengalaman Belajar Imersif

Untuk meningkatkan keterlibatan siswa, aplikasi ini juga dilengkapi dengan sebuah game edukasi interaktif yang membawa pemain seolah-olah kembali ke Indonesia pada masa lampau. Dalam game ini, siswa dapat:

- Memerankan karakter tertentu tergantung dengan chapter yang dipilih
- Mengalami langsung suasana Indonesia pada zaman penjajahan, kerajaan Nusantara, atau era pergerakan nasional melalui desain latar, musik, dan misi permainan
- Menyelesaikan berbagai misi edukatif seperti membantu pahlawan, mengumpulkan informasi sejarah, membuat keputusan penting, atau menyelesaikan teka-teki berdasarkan fakta sejarah.
- Berinteraksi dengan tokoh-tokoh sejarah yang digambarkan dalam bentuk karakter animasi, sehingga proses pembelajaran menjadi lebih personal dan berkesan.

Gameplay ini memberikan pengalaman “belajar sambil bermain” yang membuat siswa merasa menjadi bagian dari perjalanan sejarah, bukan sekadar menghafalkan fakta.

BAB II

PEMBAHASAN

2.1 Fitur Program

2.1.1 Fitur pada Aplikasi Utama

Aplikasi utama berfungsi sebagai antarmuka awal sebelum pengguna memasuki game. Beberapa fitur yang disediakan antara lain:

1. Tampilan Desain Bertema Sejarah

Aplikasi menampilkan desain visual yang memuat elemen-elemen sejarah, seperti tokoh sejarahwan, pahlawan nasional, dan simbol-simbol yang relevan dengan masa penjajahan. Tujuannya adalah memberikan kesan edukatif sebelum pengguna mulai bermain.

2. Fitur Menonton Video Sejarah

Pengguna dapat mengakses dan menonton beberapa video yang berisi materi sejarah, seperti:

- Penjelasan mengenai masa penjajahan Belanda
- Kisah perjuangan para pahlawan
- Dokumentasi sejarah bangsa

Fitur ini mendukung aspek edukasi pada aplikasi.

3. Navigasi Menu ke Game

Pada bagian bawah halaman utama, terdapat tombol atau menu yang mengarahkan pengguna untuk memulai game. Area ini menampilkan informasi singkat bahwa game memiliki alur cerita bertema perjuangan pada masa penjajahan Belanda.

2.1.2 Fitur pada Game Utama

Game yang dikembangkan merupakan game aksi dengan alur cerita kerajaan Nusantara yang sedang menghadapi serangan tentara Belanda. Fitur-fitur utama game adalah sebagai berikut:

1. Cerita dan Latar Game

Game disertai narasi mengenai situasi kerajaan pada masa penjajahan. Latar tempat dan karakter disesuaikan dengan era kolonial, sehingga pemain dapat merasakan nuansa sejarah dalam permainan.

2. Kontrol dan Aksi Player

Player memiliki beberapa kemampuan utama yang dapat digunakan untuk bertahan dan menyerang musuh, yaitu:

a. Serangan Utama (Left Click)

Pemain dapat melakukan serangan utama menggunakan tombol klik kiri mouse. Serangan ini menjadi kemampuan dasar untuk menghadapi musuh.

b. Menggunakan Perisai (Right Click)

Klik kanan digunakan untuk mengaktifkan perisai. Perisai berfungsi menahan serangan musuh sehingga pemain dapat mengurangi damage yang diterima.

c. Dash / Menghindar (Spacebar)

Menekan tombol spasi membuat karakter melakukan dash. Fitur ini digunakan untuk menghindari serangan musuh atau berpindah posisi dengan cepat.

d. Healing (Tombol H)

Pemain dapat memulihkan darah (HP) dengan menekan tombol H. Fitur ini memberikan kesempatan bertahan lebih lama dalam pertempuran.

3. Musik dan Efek Suara (SFX)

Seluruh permainan diiringi dengan:

- Background music untuk menciptakan suasana intens dalam game.
 - Sound effect untuk memperkuat aksi seperti:
 - Suara pukulan saat menyerang
 - Efek suara perisai
 - Suara heal
- Semua ini menambah imersi dan pengalaman bermain.

4. Musuh Utama

Game memiliki dua tipe musuh:

a. Musuh Biasa: Tentara Belanda

Musuh yang akan dihadapi pemain sepanjang perjalanan. Mereka mewakili kekuatan penjajah yang harus dilawan.

b. Boss Akhir

Di akhir permainan, pemain dihadapkan pada boss yang memiliki kekuatan lebih besar dari musuh biasa. Mengalahkan boss menjadi syarat utama untuk menyelesaikan game.

5. Sistem Kalah dan Menang

- Game Over — Muncul ketika darah pemain (HP) habis. Pemain akan diarahkan untuk mengulang permainan.
- Victory Scene — Ditampilkan jika pemain berhasil mengalahkan boss. Scene ini menjadi tanda bahwa pemain telah menyelesaikan seluruh tantangan dalam game.

2.2 Penjelasan Code

1. Aplikasi

Main.java

```
package com.game.aplikasigame;

import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.stage.Stage;
import javafx.util.Duration;

import java.io.IOException;
import java.lang.reflect.Method;
import java.util.List;
```

Pertama, program melakukan import pada library javafx untuk membuat GUI pada program, library List dari java.util untuk menyimpan beberapa data dengan tipe tertentu, dan Method dari java.lang.reflect fitur yang memungkinkan program melihat dan memanipulasi struktur dirinya sendiri saat runtime.

```
public class Main extends Application { + 1 usage

    private static final String GAME_LAUNCHER_CLASS = "com.gdx.lwjgl3.Lwjgl3Launcher"; 2 usages

    List<String> videoList = List.of( 4 usages
        "/video/video1.mp4",
        "/video/video2.mp4",
        "/video/video3.mp4"
    );
}
```

Selanjutnya pada class Main, program terlebih dahulu mendeklarasikan variabel GAME_LAUNCHER_CLASS yang digunakan untuk menyimpan string perintah untuk menjalankan game nantinya. Lalu program mendeklarasikan sebuah

List dari string yang menyimpan string dari lokasi video yang nantinya akan ditampilkan di program.

```
@Override
public void start(Stage stage) throws IOException {
    // ===== Bagian Header / Judul =====
    ImageView imageView = new ImageView(
        new Image(getClass().getResource(name: "/images/judul.png").toExternalForm())
    );
    imageView.setFitWidth(600);
    imageView.setPreserveRatio(true);

    Label captionZejara = new Label(s: "Belajar sejarah dengan cara interaktif dan seru!");
    captionZejara.getStyleClass().add("captionZejara");
    captionZejara.setTranslateX(-30);

    Button buttonExplore = new Button(s: "Jelajah!");
    buttonExplore.getStyleClass().add("buttonExplore");
    buttonExplore.setTranslateX(140);
    buttonExplore.setTranslateY(80);

    VBox judulZejaraBox = new VBox(imageView, captionZejara, buttonExplore);
    judulZejaraBox.setTranslateY(500);
    judulZejaraBox.setTranslateX(1200);

    Pane mainBox = new Pane(judulZejaraBox);
    mainBox.getStyleClass().add("mainBox");

    Image bg = new Image(getClass().getResource(name: "/images/bg.png").toExternalForm());
    BackgroundImage backgroundImage = new BackgroundImage(
        bg,
        BackgroundRepeat.NO_REPEAT,
        BackgroundRepeat.NO_REPEAT,
        BackgroundPosition.CENTER,
        new BackgroundSize(BackgroundSize.AUTO, BackgroundSize.AUTO, b: false, b1: false, b2: true, b3: true)
    );
    mainBox.setBackground(new Background(backgroundImage));
    mainBox.setMinHeight(1200);
```

Kode diatas merupakan kode untuk halaman pertama dari program. Pertama program menginisialisasi variabel imageView untuk menampilkan gambar judul dari program pada halaman pertama. Kemudian captionSejarah merupakan variabel untuk menampilkan caption dari judul.

Kemudian program menambahkan buttonExplore yang ketika di tekan maka program akan lanjut ke halaman berikutnya. Fitur-fitur seperti judul, gambar, dan button disimpan pada variabel judulZejaraBox yang nantinya di simpan pada mainBox. Latar belakang dari mainBox juga diubah agar terlihat lebih menarik.

```
// ===== Halaman 2: Video =====
ImageView thumbnailView = new ImageView();
thumbnailView.setFitWidth(700);
thumbnailView.setFitHeight(400);
thumbnailView.setPreserveRatio(false);
thumbnailView.setImage(new Image(getClass().getResource(name: "/thumbnail/video1.png").toExternalForm()));

final int[] currentIndex = {0};
final MediaPlayer[] currentPlayer = {null};
MediaView mediaView = new MediaView();
mediaView.setFitWidth(700);
mediaView.setFitHeight(400);

Button btnPlay = new Button(s: "Play");
btnPlay.setStyle("-fx-padding: 10px;");
btnPlay.getStyleClass().add("btnPlay");

Runnable loadVideo = () -> {
    try {
        String thumbPath = "/thumbnail/video" + (currentIndex[0] + 1) + ".png";
        thumbnailView.setImage(new Image(getClass().getResource(thumbPath).toExternalForm()));

        if (currentPlayer[0] != null) {
            currentPlayer[0].stop();
        }

        Media media = new Media(getClass().getResource(videoList.get(currentIndex[0])).toExternalForm());
        MediaPlayer newPlayer = new MediaPlayer(media);
        newPlayer.setAutoPlay(false);

        mediaView.setMediaPlayer(newPlayer);
        currentPlayer[0] = newPlayer;

        newPlayer.setOnReady(() -> {
            thumbnailView.setVisible(false);
            mediaView.setVisible(true);
        });

        thumbnailView.setVisible(true);
        mediaView.setVisible(false);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

```

loadVideo.run();

Button btnPrevVid = new Button(s: "<");
btnPrevVid.setStyle("-fx-font-size: 30px; -fx-padding: 20px;");
btnPrevVid.getStyleClass().add("btnPrevVid");

Button buttonNextVid = new Button(s: ">");
buttonNextVid.setStyle("-fx-font-size: 30px; -fx-padding: 20px;");
buttonNextVid.getStyleClass().add("buttonNextVid");

btnPrevVid.setOnAction(ActionEvent e -> {
    currentIndex[0] = (currentIndex[0] - 1 + videoList.size()) % videoList.size();
    loadVideo.run();
});

buttonNextVid.setOnAction(ActionEvent e -> {
    currentIndex[0] = (currentIndex[0] + 1) % videoList.size();
    loadVideo.run();
});

btnPlay.setOnAction(ActionEvent e -> {
    if (currentPlayer[0] != null) {
        currentPlayer[0].play();
    }
});

StackPane videoContainer = new StackPane(thumbnailView, mediaView);
videoContainer.setPrefSize(v: 700, v1: 400);
videoContainer.setMaxSize(v: 700, v1: 400);

HBox videoControls = new HBox(v: 30, btnPrevVid, videoContainer, buttonNextVid);
videoControls.setAlignment(Pos.CENTER);

VBox mainBox2 = new VBox(v: 40, videoControls, btnPlay);
mainBox2.setAlignment(Pos.CENTER);
mainBox2.setMinHeight(1200);

```

Selanjutnya program mulai pada bagian halaman kedua yang menampilkan video pembelajaran sejarah. pertama program menginisialisasi thumbnailView untuk menampilkan thumbnail video dari video yang sedang ditampilkan. Kemudian program menambahkan button play yang jika di klik video akan dimulai dan tombol kanan dan kiri yang jika di klik video akan berubah ke video selanjutnya. Apabila video masih di load, maka yang ditampilkan adalah thumbnail, namun apabila video sudah di load maka program akan menampilkan videonya. Fitur video dan gambar thumbnail akan disimpan di videoContainer dan fitur button, dan videoContainer akan

disimpan di videoControls. Lalu semua fitur pada halaman 2 akan disimpan pada mainBox2.

```
Image bg2 = new Image(getClass().getResource( name: "/images/bg2.png").toExternalForm());
BackgroundImage backgroundImage2 = new BackgroundImage(
    bg2,
    BackgroundRepeat.NO_REPEAT,
    BackgroundRepeat.NO_REPEAT,
    BackgroundPosition.CENTER,
    new BackgroundSize(BackgroundSize.AUTO, BackgroundSize.AUTO, b: false, b1: false, b2: true, b3: true)
);
mainBox2.setBackground(new Background(backgroundImage2));
```

Kemudian program juga mengubah latar belakang dari mainBox2 agar lebih menarik.

```
// ===== Halaman 3: Cards & Mainkan button =====
VBox mainBox3 = new VBox();
mainBox3.setAlignment(Pos.CENTER);
mainBox3.setMinHeight(1200);

Image bg3 = new Image(getClass().getResource( name: "/images/bg3.png").toExternalForm());
BackgroundImage backgroundImage3 = new BackgroundImage(
    bg3,
    BackgroundRepeat.NO_REPEAT,
    BackgroundRepeat.NO_REPEAT,
    BackgroundPosition.CENTER,
    new BackgroundSize(BackgroundSize.AUTO, BackgroundSize.AUTO, b: false, b1: false, b2: true, b3: true)
);
mainBox3.setBackground(new Background(backgroundImage3));

VBox card1 = new VBox( v: 20);
card1.setAlignment(Pos.CENTER);
card1.setStyle("-fx-background-color: rgba(255,255,255,0.7); -fx-padding: 20px; -fx-background-radius: 20px;");
card1.setPrefSize( v: 400, v1: 500);

ImageView img1 = new ImageView(new Image(getClass().getResource( name: "/images/mataramislam.jpg").toExternalForm()));
img1.setFitWidth(300);
img1.setPreserveRatio(true);
Label text1 = new Label( s: "Kerajaan Mataram Islam didirikan pada akhir abad ke-16\n oleh Sutawijaya (Panembahan Senopati) \n di K");
text1.getStyleClass().add("text1");
card1.getChildren().addAll(img1, text1);

VBox card2 = new VBox( v: 20);
card2.setAlignment(Pos.CENTER);
card2.setStyle("-fx-background-color: rgba(255,255,255,0.7); -fx-padding: 20px; -fx-background-radius: 20px;");
card2.setPrefSize( v: 400, v1: 500);

ImageView img2 = new ImageView(new Image(getClass().getResource( name: "/images/kembar.png").toExternalForm()));
img2.setFitWidth(300);
img2.setPreserveRatio(true);
Label text2 = new Label( s: "Chapter 1 : Si Kembar dari Mataram");
text2.getStyleClass().add("text2");

VBox card3 = new VBox( v: 20);
card3.setAlignment(Pos.CENTER);
card3.setStyle("-fx-background-color: rgba(255,255,255,0.7); -fx-padding: 20px; -fx-background-radius: 20px;");
card3.setPrefSize( v: 400, v1: 500);
```

```
btnPrevCard.setOnAction( ActionEvent e -> {
    currentCard[0] = (currentCard[0] - 1 + cards.size()) % cards.size();
    loadCard.run();
});

btnNextCard.setOnAction( ActionEvent e -> {
    currentCard[0] = (currentCard[0] + 1) % cards.size();
    loadCard.run();
});
```

Selanjutnya program memasuki halaman 3, pertama program menginisialisasi mainBox3 untuk menampung semua fitur nantinya. Kemudian latar belakang mainBox3 diubah agar lebih menarik. Pertama kita membuat card1 yang akan menampilkan gambar dan cerita dari kerajaan Mataram Islam, card2 akan menampilkan game sejarah mengenai Mataram Islam dan juga terdapat button play yang jika ditekan maka game akan dijalankan. Kemudian card3 adalah game selanjutnya yang masih dalam tahap pembangunan sehingga hanya menampilkan gambar Coming Soon! tanpa bisa dimulai. Terdapat juga button untuk menggeser ke kanan dan ke kiri untuk melihat card lain.

```
// ===== Container & Scene =====
VBox container = new VBox(mainBox, mainBox2, mainBox3);
ScrollPane scrollPane = new ScrollPane(container);
scrollPane.setFitToWidth(true);

Scene scene = new Scene(scrollPane, v: 1080, v1: 1080);
scene.getStylesheets().add(getClass().getResource( name: "/css/style.css").toExternalForm());

// tombol explore (scroll)
buttonExplore.setOnAction( ActionEvent e -> {
    Timeline scrollAnim = new Timeline(
        new KeyFrame(Duration.millis( v: 0),
            new KeyValue(scrollPane.vvalueProperty(), scrollPane.getVvalue())
        ),
        new KeyFrame(Duration.millis( v: 500),
            new KeyValue(scrollPane.vvalueProperty(), t: 0.5)
        )
    );
    scrollAnim.play();
});

// ===== ACTION: Mainkan! -> jalankan game LWJGL3 =====
detailBtn.setOnAction( ActionEvent e -> {
    // 1) tutup / sembunyikan launcher
    Stage win = (Stage) detailBtn.getScene().getWindow();
    try {
        // prefer close supaya windowing resources dilepas
        win.close();
    } catch (Exception ignore) {
        win.hide();
    }
});
```

```

// 2) jalankan game di thread terpisah (penting)
new Thread(() -> {
    try {
        // Pemanggilan via refleksi agar mudah ganti FOCN tanpa recompile
        Class<?> cls = Class.forName(GAME_LAUNCHER_CLASS);
        Method main = cls.getMethod( name: "main", String[].class);
        // panggil static main(new String[]{});
        main.invoke( obj: null, (Object) new String[]{} );
    } catch (ClassNotFoundException cnf) {
        System.err.println("Game launcher class not found: " + GAME_LAUNCHER_CLASS);
        cnf.printStackTrace();
    } catch (Throwable t) {
        t.printStackTrace();
    }
}, name: "Game-Thread").start();
});

// show stage
stage.setTitle("Game Launcher");
stage.setScene(scene);
stage.setFullScreen(true);
stage.show();
}

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

```

Selanjutnya pada bagian akhir dari program, program akan menampung semua mainBox pada container yang nantinya akan ditampung lagi pada scrollPane agar program bisa di scroll. Selanjutnya program memberikan perintah agar jika button explore ditekan maka program akan scroll ke halaman ke 2. Jika button Mainkan ditekan, maka program akan melakukan perintah untuk menjalankan game. Terakhir program akan menampilkan program dengan full screen.

2. Game

Main.java

```
package com.gdx;

import com.badlogic.gdx.Game;

public class Main extends Game { 8 usages
    @Override
    public void create() {
        setScreen(new IntroScreen( game: this));
    }
}
```

Kode program ini mendefinisikan kelas Main yang berada dalam paket com.gdx dan berfungsi sebagai inti pengontrol alur permainan (Game Lifecycle). Kelas ini mewarisi (extends) kelas abstrak Game dari pustaka LibGDX (com.badlogic.gdx.Game). Penggunaan kelas Game ini sangat krusial karena memungkinkan aplikasi untuk memanajemen perpindahan antar layar (screens) secara dinamis, seperti berpindah dari menu utama ke level permainan, berbeda dengan ApplicationListener standar yang cenderung statis.

Di dalam kelas ini, terdapat metode create() yang merupakan metode pertama yang dipanggil oleh framework saat aplikasi berhasil dimuat. Pada implementasi ini, saya menggunakan perintah setScreen(new IntroScreen(this));. Baris kode ini bertujuan untuk langsung menginisialisasi dan menampilkan layar pembuka atau IntroScreen begitu aplikasi dijalankan. Parameter this yang dilewatkan ke dalam konstruktor IntroScreen mereferensikan objek Main itu sendiri, yang bertujuan agar kelas IntroScreen nantinya memiliki akses untuk mengubah layar (switching screens) ke tahapan permainan selanjutnya.

NPC.java

```
package com.gdx;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.scenes.scene2d.Actor;
```

Bagian ini merupakan langkah awal inisialisasi modul. Saya mendefinisikan paket tempat kelas ini berada, yaitu com.gdx. Selanjutnya, saya melakukan impor terhadap pustaka-pustaka LibGDX yang esensial. Texture dan TextureRegion digunakan untuk memanipulasi aset gambar, Animation untuk mengatur perpindahan frame gerak, Batch sebagai media rendering ke GPU, Rectangle untuk logika fisika (area sentuh hitbox), dan Actor sebagai kelas induk yang memungkinkan NPC ini dikelola dalam sebuah Stage.

```
public class NPC extends Actor { 2 usages

    public enum State { 8 usages
        ENTERING, 4 usages
        WAITING, 2 usages
        LEAVING, 4 usages
        IDLE 3 usages
    }

    public State state = State.IDLE; 11 usages
    Animation<TextureRegion> walkAnim; 2 usages
    TextureRegion idleFrame; 2 usages
    TextureRegion[] walkFrames; 3 usages
    public Rectangle hitbox; 3 usages
    public float speed = 100; 2 usages
    float stateTime = 0f; 4 usages
    float scale = 0.25f; 4 usages
    float stopX = 100; // where NPC stops for dialog 2 usages
```

Pada blok ini, saya mendefinisikan kelas NPC yang mewarisi sifat-sifat (extends) dari kelas Actor. Untuk mengatur perilaku NPC, saya menerapkan konsep

Finite State Machine sederhana menggunakan enum State dengan empat kondisi: ENTERING (masuk), WAITING (diam), LEAVING (keluar), dan IDLE (non-aktif). Variabel-variabel instan juga dideklarasikan di sini, termasuk objek animasi, frame gambar, hitbox untuk deteksi kolisi, kecepatan gerak (speed), skala gambar (scale), serta stopX yang menentukan titik koordinat di mana NPC harus berhenti untuk memulai dialog.

```
public NPC(float x, float y, Texture npcSheet) { 1 usage
    setPosition(x, y);

    // sprite sheet layout: 2 rows x 7 cols
    int cols = 7;
    int rows = 2;
    int frameW = npcSheet.getWidth() / cols;
    int frameH = npcSheet.getHeight() / rows;
    TextureRegion[][] tmp = TextureRegion.split(npcSheet, frameW, frameH);

    // row 1 = walk frames, row 0 frame 0 = idle
    walkFrames = new TextureRegion[cols];
    for (int i = 0; i < cols; i++) walkFrames[i] = tmp[1][i];
    idleFrame = tmp[0][0];

    walkAnim = new Animation<TextureRegion>(frameDuration: 0.12f, walkFrames);

    // hitbox sized from raw frame dims and scale
    hitbox = new Rectangle(getX(), getY(), width: frameW * scale, height: frameH
}
```

Di dalam konstruktor, saya melakukan inisialisasi posisi awal dan pemrosesan aset grafis. Aset gambar (npcSheet) yang berupa sprite sheet dipotong-potong menjadi grid menggunakan TextureRegion.split berdasarkan jumlah kolom (7) dan baris (2). Baris kedua (indeks 1) diambil sebagai kumpulan frame animasi berjalan (walkFrames), sedangkan baris pertama frame awal diambil sebagai pose diam (idleFrame). Animasi diatur dengan durasi 0.12 detik per frame. Terakhir, saya membentuk hitbox bertipe Rectangle yang ukurannya disesuaikan dengan dimensi frame dikalikan skala agar presisi.

```
public void startEnter() { state = State.ENTERING; stateTime = 0f; }
public void startLeave() { state = State.LEAVING; stateTime = 0f; }
```

Dua metode sederhana ini berfungsi sebagai pemicu perubahan perilaku NPC dari luar kelas (misalnya dipanggil oleh event manager). startEnter() mengubah status menjadi ENTERING untuk memulai animasi masuk, sedangkan startLeave() mengubah status menjadi LEAVING untuk menyuruh NPC pergi. Penting dicatat

bahwa saya mereset stateTime menjadi 0 setiap kali status berubah agar animasi dimulai ulang dari frame pertama.

```
@Override
public void act(float delta) {
    super.act(delta);
    stateTime += delta;
    switch (state) {
        case ENTERING:
            moveBy( x: speed * delta, y: 0 );
            if (getX() >= stopX) { setX(stopX); state = State.WAITING; }
            break;
        case WAITING:
            break;
        case LEAVING:
            moveBy( x: -speed * delta, y: 0 );
            if (getX() <= -300) { setX(-300); state = State.IDLE; }
            break;
        case IDLE:
            break;
    }
    hitbox.setPosition(getX(), getY());
}
```

Metode act adalah jantung logika yang dijalankan setiap frame (game loop). Saya melakukan override metode ini untuk menghitung akumulasi waktu (stateTime) dan mengeksekusi logika pergerakan berdasarkan state saat ini menggunakan switch-case. Jika status ENTERING, NPC bergerak ke kanan hingga mencapai titik stopX lalu berubah menjadi WAITING. Jika status LEAVING, NPC bergerak ke kiri hingga keluar layar (koordinat -300) lalu menjadi IDLE. Di akhir metode, posisi hitbox diperbarui agar selalu mengikuti posisi visual NPC.

```
@Override
public void draw(Batch batch, float parentAlpha) {
    TextureRegion frame;
    if (state == state.ENTERING || state == State.LEAVING) {
        frame = walkAnim.getKeyFrame(stateTime, looping: true);
    } else {
        frame = idleFrame;
    }
    TextureRegion r = new TextureRegion(frame);
    if (state == state.ENTERING) {
        if (r.isFlipX()) r.flip(x: true, y: false);
    }
    if (state == State.LEAVING) {
        if (!r.isFlipX()) r.flip(x: true, y: false);
    }
    batch.draw(r, getX(), getY(), v2: r.getRegionWidth() * scale, v3: r.getRegionHeight());
}
```

Metode draw bertugas menggambar NPC ke layar. Di sini saya menentukan frame mana yang harus ditampilkan: jika sedang bergerak (ENTERING/LEAVING), ambil frame dari walkAnim, jika tidak, ambil idleFrame. Saya menggunakan TextureRegion r sebagai penampung sementara agar manipulasi orientasi gambar (flip) tidak merusak aset asli. Logika flip memastikan NPC menghadap ke kanan saat masuk dan menghadap ke kiri saat keluar. Terakhir, gambar dirender menggunakan batch.draw dengan koordinat dan skala yang telah ditentukan.

```
public Rectangle getHitbox() { return hitbox; } no usages
```

Metode terakhir ini hanyalah sebuah getter untuk mengembalikan objek hitbox. Metode ini dipersiapkan agar kelas lain (misalnya kelas yang menangani deteksi sentuhan input user atau tabrakan peluru) bisa mendapatkan area persegi yang merepresentasikan fisik NPC tersebut.

Bullet.java

```
package com.gdx;

import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.scenes.scene2d.Actor;
```

Tahap pertama adalah mendefinisikan paket dan mengimpor pustaka yang dibutuhkan dari framework LibGDX. Saya menggunakan Batch dan TextureRegion untuk keperluan rendering grafis, Rectangle untuk membuat area fisik (hitbox) guna deteksi tabrakan, dan Actor sebagai kelas induk agar peluru ini dapat dimasukkan ke dalam panggung permainan (Stage) dan memiliki siklus hidup yang terkelola.

```
public class Bullet extends Actor { 9 usages
    private TextureRegion region; 2 usages
    public Rectangle rect; 11 usages
    public float velX, velY; 3 usages
    public float rotation = 0f; 2 usages
    private Player target; 8 usages
    private int worldWidth = 5000; 1 usage
    private int worldHeight = 2000; 1 usage
```

Kelas Bullet didefinisikan sebagai turunan dari Actor. Di sini, saya mendeklarasikan variabel-variabel properti peluru. TextureRegion region menyimpan gambar peluru, sedangkan Rectangle rect berfungsi sebagai hitbox fisiknya. Variabel velX dan velY menentukan arah dan kecepatan peluru. Saya juga menyimpan referensi ke objek Player dalam variabel target untuk memvalidasi tabrakan. Selain itu, worldWidth dan worldHeight ditetapkan sebagai batas dunia permainan untuk menentukan kapan peluru harus dihapus jika tidak mengenai sasaran.

```
public Bullet(TextureRegion region, float x, float y, float w, float h, float vX, float vY, Player target) {
    this.region = region;
    setBounds(x, y, w, h);
    rect = new Rectangle(x, y, w, h);
    this.velX = vX;
    this.velY = vY;
    this.target = target;
}
```

Pada konstruktor, saya melakukan inisialisasi parameter yang diterima saat peluru dibuat (diinstansiasi). Parameter tersebut meliputi gambar tekstur, posisi awal (x, y), dimensi (w, h), kecepatan vektor (vX, vY), dan target pemain. Metode setBounds dipanggil untuk menetapkan batas visual aktor, sementara objek rect diinstansiasi sebagai batas logika fisik yang akan mengikuti posisi visual tersebut.

```
@Override  
public void act(float delta) {  
    super.act(delta);  
    // update posisi  
    moveBy(x: velX * delta, y: velY * delta);  
    rect.setPosition(getX(), getY());  
  
    // tabrak player  
    if (target != null && rect.overlaps(target.getHitbox())) {  
        // jika player shield aktif, peluru hilang tanpa damage  
        if (!target.isShielding && target.health > 0) {  
            target.health -= 10;  
            if (target.health < 0) target.health = 0;  
        }  
        remove(); // hapus diri  
        return;  
    }  
  
    // keluar batas world -> hapus  
    if (getX() < 0 || getX() > worldWidth || getY() < 0 || getY() > worldHeight) {  
        remove();  
    }  
}
```

Metode act adalah inti dari logika peluru yang dijalankan setiap frame. pertama, saya memperbarui posisi peluru berdasarkan kecepatannya (velX, velY) dikalikan delta time agar pergerakan halus, lalu menyinkronkan posisi rect (hitbox). Kedua, saya melakukan deteksi tabrakan. Jika peluru menyentuh hitbox target (Player), sistem akan mengecek apakah pemain sedang menggunakan perisai (isShielding). Jika tidak sedang shielding dan darah masih ada, darah pemain dikurangi 10 poin. Peluru kemudian dihapus (remove()) setelah menabrak. Ketiga, saya menambahkan pengecekan batas dunia. Jika peluru meleset dan keluar dari koordinat area permianan (worldWidth atau worldHeight), peluru dihapus dari memori untuk mencegah kebocoran kinerja.

```
@Override  
public void draw(Batch batch, float parentAlpha) { batch.draw(region, getX(), getY(), getWidth(), getHeight()); }  
  
public Rectangle getRect() { return rect; }
```

Terakhir, metode draw bertugas menggambar tekstur peluru ke layar menggunakan Batch pada posisi dan dimensi yang telah diupdate. Saya juga

menyediakan metode getRect() yang bersifat publik untuk memberikan akses ke data hitbox peluru, yang mungkin diperlukan oleh kelas lain (misalnya Debug Renderer atau manajer kolisi eksternal).

Boss.java

```
class Bomb { 6 usages
    public Rectangle rect; 17 usages
    public TextureRegion sprite; 4 usages
    public boolean exploded = false; 3 usages
    public float velX; 2 usages
    public float velY = -200f; // jatuh ke bawah 1 usage

    public Bomb(float x, float y, float targetX, TextureRegion tex) { 1 usage
        rect = new Rectangle(x, y, width: 50, height: 50);
        sprite = tex;
        float speedX = 200f;
        velX = (targetX > x) ? speedX : -speedX;
    }

    public void update(float delta) { 1 usage
        rect.x += velX * delta;
        rect.y += velY * delta;
        if (rect.y <= 0) {
            rect.y = 0;
            exploded = true;
        }
    }

    public void draw(Batch batch) { no usages
        if (sprite != null) batch.draw(sprite, rect.x, rect.y, rect.width, rect.height);
    }
}
```

Kelas Bomb dirancang sebagai proyektil dengan lintasan parabolik sederhana. Pada konstruktor, saya menentukan arah horizontal (velX) berdasarkan posisi target relatif terhadap posisi awal bom; jika target berada di kanan, kecepatan positif, dan sebaliknya. Variabel velY diinisialisasi dengan nilai negatif (-200f) untuk mensimulasikan gravitasi konstan. Dalam metode update, posisi diperbarui setiap frame berdasarkan waktu delta. Ketika bom menyentuh tanah (koordinat y <= 0), variabel exploded diubah menjadi true sebagai penanda bagi sistem utama untuk memicu ledakan dan menghapus objek bom ini.

```
class Explosion { 5 usages
    public boolean hasDamaged = false; 2 usages
    public float x, y; 3 usages
    public float life = 0.3f; 4 usages
    public float radius = 100f; 1 usage

    float stateTime = 0f; 2 usages
    Animation<TextureRegion> anim; 2 usages

    public Explosion(float x, float y, Animation<TextureRegion> anim) { 2 usages
        this.x = x;
        this.y = y;
        this.anim = anim;
    }

    public void update(float delta) { 2 usages
        stateTime += delta;
        life -= delta;
    }

    public TextureRegion getFrame() { 1 usage
        return anim.getKeyFrame(stateTime, looping: false);
    }
}
```

Kelas Explosion berfungsi sebagai efek visual yang bersifat sementara. Saya menetapkan variabel life sebesar 0.3 detik yang berfungsi sebagai durasi hidup objek ini sebelum dihapus dari memori. Logika animasi diatur menggunakan stateTime yang terus bertambah di setiap pemanggilan update, yang kemudian digunakan oleh metode getFrame() untuk mengambil frame animasi yang tepat dari objek Animation LibGDX tanpa looping (mode false pada getKeyFrame).

```

public class Boss extends Actor { 3 usages
    public float hp = 500f; 3 usages
    public float maxHp = 500f; 1 usage

    public boolean active = false; 7 usages
    public Rectangle hitbox; 7 usages

    public enum State { IDLE, RUN, ROLL } 7 usages
    public State state = State.IDLE; 6 usages

    TextureRegion idleFrame; 2 usages
    Animation<TextureRegion> runAnim; 2 usages
    Animation<TextureRegion> rollAnim; 2 usages

    private float stateTime = 0f; 6 usages
    private float speed = 150f; 2 usages
    private boolean facingRight = true; 9 usages

    // Bomb & Explosion
    private float bombCooldown = 0f; 4 usages
    private float bombCooldownMax = 5f; 1 usage
    private TextureRegion bombTex; 2 usages
    private Animation<TextureRegion> explosionAnim; 2 usages

    // Roll mechanics
    private float rollCooldown = 0f; 4 usages
    private float rollCooldownMax = 12f; 1 usage
    private float rollSpeed = 450f; // tweak supaya terasa dash 1 usage
    private boolean rolling = false; 4 usages
    private int rollDirection = 1; 4 usages
    private float rollDistanceLeft = 0f; 5 usages
    private boolean rollHasDamaged = false; 2 usages

```

Kelas Boss dibangun sebagai entitas kompleks yang mewarisi Actor. Saya menerapkan Finite State Machine (FSM) dengan tiga kondisi utama: IDLE, RUN, dan ROLL untuk mengatur perilaku. Pada konstruktor, dilakukan pemotongan tekstur (texture splitting) secara dinamis untuk membuat animasi lari (4 frame) dan menggelinding (8 frame). Selain itu, saya juga mendefinisikan hitbox yang lebih kecil daripada ukuran tekstur asli ($w * 0.45f, h * 0.55f$) agar deteksi tabrakan (collision detection) terasa lebih adil dan presisi bagi pemain.

```

@Override
public void act(float delta) {
    super.act(delta);
    if (!active) return;
    // RESETABLE DAMAGE COOLDOWN
    if (dmgCooldown > 0)
        dmgCooldown -= delta;

    // Jika sedang roll: prioritaskan roll (isolasi perilaku)
    if (rolling) {
        stateTime += delta;
        bombCooldown -= delta;
        rollCooldown -= delta;
        doRoll(delta);
        updateHitbox();
        clampPosition();
        return; // penting: hentikan logic lain selama roll
    }

    // Normal flow
    stateTime += delta;
    bombCooldown -= delta;
    rollCooldown -= delta;

    Player p = getStage().getRoot().findActor("player");
    if (p == null) return;

    float bossCenter = getX() + getWidth() / 2f;
    float targetCenter = p.getX() + p.getWidth() / 2f;
    facingRight = targetCenter > bossCenter;
}

```

Metode `act` merupakan inti dari kecerdasan buatan Boss. Logika ini memprioritaskan aksi ROLL (menggelinding) di atas segalanya; jika sedang menggelinding, logika lain dihentikan sementara (`return`). Jika dalam kondisi normal, Boss akan mendeteksi jarak pemain. Jika jarak > 300 unit, Boss akan mengejar (`chasePlayer`). Jika jarak sudah dekat, Boss memasuki mode menyerang dengan probabilitas acak menggunakan `MathUtils.randomBoolean`. Prioritas serangan diatur mulai dari melempar bom, melakukan dash/roll, hingga menembak secara burst (beruntun).

```
private void startRoll(float targetCenter, float bossCenter) { 1 usage
    rolling = true;
    state = State.ROLL;
    rollDirection = (targetCenter > bossCenter) ? 1 : -1;
    rollDistanceLeft = 420f;
    rollCooldown = rollCooldownMax;

    // sinkron animasi & orientasi supaya roll terlihat benar
    stateTime = 0f;
    facingRight = (rollDirection == 1);
    rollHasDamaged = false;
}
```

```
private void doRoll(float delta) { 1 usage
    float move = rollSpeed * delta;
    if (move > rollDistanceLeft) move = rollDistanceLeft;
    setX(getX() + move * rollDirection);
    rollDistanceLeft -= move;

    if (rollDistanceLeft <= 0f) {
        rolling = false;
        state = State.IDLE;
        rollHasDamaged = false;
        updateHitbox(); // kembalikan hitbox normal
    }
    clampPosition();
}
```

```
private void spawnShoot(Player target) { 2 usages
    float pistolOffsetX = getWidth() * 0.14f;
    float pistolHeightY = getHeight() * 0.22f;
    float spawnX = getX() + getWidth() / 2f;
    spawnX += facingRight ? pistolOffsetX : -pistolOffsetX;
    float spawnY = getY() + pistolHeightY + MathUtils.random(-8f, 8f);
    float vel = facingRight ? 380f : -380f;

    if (shootSound != null) shootSound.play(0.9f);

    if (bullets != null) {
        Bullet bossBullet = new Bullet(bulletTex, spawnX, spawnY, w: 40, h: 40, vel, vy: 0, target);
        bossBullet.rotation = facingRight ? 0 : 180;
        bullets.add(bossBullet);
        getStage().addActor(bossBullet);
    }
}
```

Metode startRoll dan doRoll menangani mekanisme gerakan cepat (dash). Saya menggunakan variabel rollDistanceLeft untuk memastikan Boss bergerak sejauh 420 unit secara konsisten. Selama roll, kontrol penuh diambil alih oleh fungsi ini. Sementara itu, spawnShoot menangani instansiasi objek Bullet. Posisi spawn peluru dihitung secara dinamis berdasarkan arah hadap Boss (facingRight) agar peluru keluar seolah-olah dari senjata, bukan dari tengah tubuh. Saya juga menambahkan sedikit variasi acak pada sumbu Y (MathUtils.random(-8f, 8f)) agar tembakan tidak terlalu linear.

```
@Override
public void draw(Batch batch, float parentAlpha) {
    TextureRegion frame;
    switch (state) {
        case ROLL: frame = rollAnim.getKeyFrame(stateTime, looping: true); break;
        case RUN: frame = runAnim.getKeyFrame(stateTime, looping: true); break;
        default:
            float breathe = MathUtils.sin( radians: stateTime * 4f) * 2f;
            frame = idleFrame;
            TextureRegion r = new TextureRegion(frame);
            if (!facingRight && !r.isFlipX()) r.flip(x: true, y: false);
            if (facingRight && r.isFlipX()) r.flip(x: true, y: false);
            batch.draw(r, v: getX() + breathe, v1: getY() - 20f + breathe * 0.5f, getWidth(), getHeight());
            return;
    }

    TextureRegion r = new TextureRegion(frame);
    if (!facingRight && !r.isFlipX()) r.flip(x: true, y: false);
    if (facingRight && r.isFlipX()) r.flip(x: true, y: false);

    batch.draw(r, getX(), v1: getY() - 20f, getWidth(), getHeight());
}
```

Metode draw bertanggung jawab merender visual Boss ke layar. Saya menambahkan efek "bernapas" (breathing effect) saat Boss dalam keadaan IDLE menggunakan fungsi sinus (MathUtils.sin), yang memberikan sedikit pergerakan naik-turun pada koordinat gambar agar Boss terlihat hidup. Selain itu, terdapat logika flip yang memastikan tekstur selalu menghadap ke arah pemain atau arah gerakan. Penggunaan stateTime pada getKeyFrame memastikan animasi berjalan dan menggelinding dimainkan dengan mulus sesuai durasi yang ditentukan.

EndScreen.java

```

package com.gdx;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.utils.Array;
import com.badlogic.gdx.utils.viewport.ScreenViewport;

public class EndScreen implements Screen { 1 usage
    private final Main game; 3 usages
    private Stage stage; 7 usages

    // Audio
    private Music winMusic; 9 usages

    private final String[] winImages = { 2 usages
        "win1.png",
        "win2.png",
        "win3.png",
        "win4.png",
        "win5.png"
    };

    private Array<Texture> textures = new Array<>(); 2 usages
    private Image currentImage; 8 usages
    private int currentIndex = 0; 3 usages
    private float timer = 0f; 3 usages
    private boolean isTransitioning = false; 6 usages
}

```

Pada bagian awal kode, saya mendefinisikan paket dan variabel-variabel global yang dibutuhkan oleh kelas ini. Variabel yang paling krusial di sini adalah winImages yang merupakan array string berisi nama-nama file gambar untuk cerita akhir, serta textures yang bertipe `Array<Texture>` untuk menyimpan asset yang dimuat agar bisa dibersihkan (dispose) nantinya. Saya juga menyiapkan variabel winMusic untuk audio dan Stage sebagai wadah elemen antarmuka.

```
public EndScreen(Main game) { 1 usage
    this.game = game;
    stage = new Stage(new ScreenViewport());

    // SETUP AUDIO
    try {
        winMusic = Gdx.audio.newMusic(Gdx.files.internal("sound/win.ogg"));
        winMusic.setLooping(false);
        winMusic.setVolume(1.0f);
        winMusic.play();
    } catch (Exception e) {
        System.out.println("Error memuat musik win: " + e.getMessage());
    }

    loadNextImage();
}
```

Konstruktor ini adalah fungsi yang pertama kali dijalankan saat EndScreen dipanggil. Tugas utamanya ada dua: menyiapkan Stage dan memutar musik kemenangan. Saya menggunakan blok try-catch saat memuat win.ogg untuk mencegah aplikasi crash jika file audio tidak ditemukan. Setelah musik diputar, saya langsung memanggil fungsi loadNextImage() untuk menampilkan gambar pertama dari cerita.

```
private void loadNextImage() { 2 usages
    if (currentIndex >= winImages.length) {
        backToIntro();
        return;
    }

    Texture tex = new Texture(winImages[currentIndex]);
    textures.add(tex);

    currentImage = new Image(tex);
    currentImage.setFillParent(true);
    currentImage.getColor().a = 0f;
    currentImage.addAction(Actions.fadeIn( duration: 1.2f));

    stage.clear();
    stage.addActor(currentImage);

    timer = 0f;
}
```

Fungsi `loadNextImage` bertanggung jawab untuk menampilkan gambar berdasarkan indeks saat ini. Pertama, fungsi ini mengecek apakah indeks sudah melebihi jumlah gambar (`winImages.length`). Jika ya, berarti cerita sudah selesai dan pemain dikembalikan ke menu awal via `backToIntro()`. Jika belum, fungsi ini memuat tekstur baru, menambahkannya ke dalam Stage, dan memberikan efek animasi `fadeIn` (muncul perlahan) selama 1.2 detik agar transisi terlihat halus.

```
@Override
public void render(float delta) {
    // Bersihkan layar dengan warna hitam
    Gdx.gl.glClearColor( v: 0, v1: 0, v2: 0, v3: 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    stage.act(delta);
    stage.draw();

    if (isTransitioning) return;

    timer += delta;

    // Ganti gambar otomatis setelah 4 detik
    if (timer > 4.0f) {
        nextImage();
    }

    // Ganti gambar manual jika user klik layar (skip)
    if (Gdx.input.justTouched() && !isTransitioning) {
        nextImage();
    }
}
```

Metode render berjalan terus-menerus setiap frame. Di sini, saya membersihkan layar dengan warna hitam dan memperbarui logika stage. Saya menerapkan logika pewaktu (timer) di mana jika waktu sudah lebih dari 4 detik, gambar akan berganti otomatis. Selain itu, saya juga menambahkan deteksi input Gdx.input.justTouched() yang memungkinkan pengguna untuk menyentuh layar jika ingin mempercepat (skip) ke gambar berikutnya tanpa menunggu timer habis.

```
private void nextImage() { 2 usages
    if (isTransitioning) return;
    isTransitioning = true;

    currentImage.addAction(Actions.sequence(
        Actions.fadeOut( duration: 1.0f),
        Actions.run(() -> {
            currentIndex++;
            isTransitioning = false;
            loadNextImage();
        })
    ));
}
```

Fungsi nextImage khusus menangani efek visual saat gambar akan diganti. Saya menggunakan Actions.sequence untuk merangkai tindakan: pertama lakukan fadeOut (gambar meredup) selama 1 detik, dan setelah itu jalankan blok kode (Actions.run) untuk menaikkan currentIndex dan memanggil loadNextImage(). Variabel isTransitioning digunakan sebagai pengunci (flag) agar fungsi ini tidak dipanggil berulang kali saat animasi sedang berjalan.

```
private void backToIntro() { 1 usage
    isTransitioning = true;

    Runnable switchScreen = () -> {
        // STOP MUSIK SEBELUM PINDAH
        if (winMusic != null && winMusic.isPlaying()) {
            winMusic.stop();
        }

        // Kembali ke Menu Awal (IntroScreen)
        game.setScreen(new IntroScreen(game));
        dispose(); // Bersihkan memori EndScreen
    };

    if (currentImage != null) {
        currentImage.addAction(Actions.sequence(
            Actions.fadeOut(duration: 0.5f),
            Actions.run(switchScreen)
        ));
    } else {
        switchScreen.run();
    }
}
```

Fungsi `backToIntro` menangani perpindahan layar dari layar akhir kembali ke layar pembuka (`IntroScreen`). Hal penting yang saya lakukan di sini adalah memastikan musik `winMusic` dihentikan (`stop`) sebelum pindah layar agar suaranya tidak bertabrakan dengan musik di menu utama. Perpindahan layar ini juga dibungkus dengan animasi `fadeOut` agar tidak terasa kasar bagi pengguna.

```
@Override  
public void resize(int width, int height) {  
    stage.setViewport().update(width, height, centerCamera: true);  
}  
  
@Override public void show() {}  
@Override public void pause() {}  
@Override public void resume() {}  
@Override public void hide() {}
```

Metode resize sangat penting untuk memastikan tampilan antarmuka (Stage) menyesuaikan diri jika ukuran jendela aplikasi berubah. stage.setViewport().update(width, height, true) memastikan elemen UI tetap proporsional. Metode-metode lain seperti show, pause, resume, dan hide dibiarkan kosong karena tidak ada logika spesifik yang diperlukan pada event-event tersebut untuk layar ini.

```
@Override  
public void resize(int width, int height) {  
    stage.setViewport().update(width, height, centerCamera: true);  
}  
  
@Override public void show() {}  
@Override public void pause() {}  
@Override public void resume() {}  
@Override public void hide() {}  
  
@Override  
public void dispose() {  
    stage.dispose();  
  
    // Dispose musik  
    if (winMusic != null) {  
        winMusic.dispose();  
    }  
  
    // Dispose semua texture yang dimuat  
    for (Texture t : textures) {  
        if (t != null) t.dispose();  
    }  
}
```

Terakhir adalah metode dispose yang wajib ada untuk mencegah kebocoran memori memory leak. Di sini, saya membersihkan stage dan objek musik winMusic. Selain itu, saya melakukan perulangan loop pada array textures untuk membersihkan semua tekstur gambar cerita yang sempat dimuat ke dalam memori. Ini adalah praktik standar dalam pengembangan game dengan LibGDX.

Enemy.java

```
✓ import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.MathUtils;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.utils.TimeUtils;
```

Pertama program melakukan import pada beberapa library dari gdx agar dapat menjalankan fungsi-fungsi seperti draw dan lain lain.

```
public class Enemy extends Actor { 10 usages ± lfunzz +1 *

    public float hp = 100f; 4 usages
    public float maxHp = 100f; 2 usages

    public boolean dead = false; 5 usages
    public float deathTime = 0f; 4 usages
    public long lastShootTime = 0; 5 usages

    public float speed = 80f; 4 usages
    public boolean facingRight = true; 8 usages

    private Rectangle hitbox; 3 usages
    private float scale = 1.0f; 1 usage

    // === ANIMASI ===
    private Animation<TextureRegion> idleAnim; 3 usages
    private Animation<TextureRegion> walkAnim; 3 usages
    private Animation<TextureRegion> runAnim; 3 usages
    private Animation<TextureRegion> shootAnim; 4 usages
    private Animation<TextureRegion> dieAnim; 4 usages

    private float stateTime = 0f; 10 usages
    private EnemyState state = EnemyState.IDLE; 17 usages
    private EnemyState prevState = EnemyState.IDLE; 1 usage

    // AI
    private float patrolTargetX; 4 usages
    private float detectDelay = 1.0f; // bisa disesuaikan 1 usage
    private float delayTimer = 0f; 2 usages
    private boolean detectedPlayer = false; 2 usages
    private float damageCooldown = 0f; 4 usages
    private boolean countedKill = false; 2 usages

    private enum EnemyState { 20 usages ± lfunzz +1
        IDLE, WALK, RUN, SHOOT, DIE 5 usages
    }

    // SHOOT timing
    private final float fireTime = 0.12f; 1 usage
    private boolean firedThisShot = false; 3 usages
```

Pada class Enemy, dilakukan extends Actor agar objek musuh dapat menggunakan fitur bawaan seperti update melalui act(), ditampilkan melalui draw(), dan dapat ditambahkan ke Stage. Enemy memiliki beberapa atribut utama yang digunakan untuk mengatur statistik, animasi, dan kecerdasan buatan musuh. Pertama terdapat hp dan maxHp yang menyimpan jumlah nyawa musuh dan batas maksimal nyawa. Variabel dead digunakan untuk menandai apakah musuh sudah mati, sedangkan deathTime menyimpan durasi animasi kematian. Atribut lastShootTime digunakan untuk mencatat kapan terakhir musuh menembak, sehingga bisa diterapkan cooldown.

Selanjutnya Enemy juga memiliki speed untuk mengatur kecepatan bergerak dan facingRight untuk menentukan apakah musuh menghadap ke kanan atau ke kiri. Hitbox disimpan menggunakan Rectangle untuk mendeteksi tabrakan, sedangkan scale digunakan untuk menentukan ukuran sprite. Pada bagian animasi, terdapat idleAnim, walkAnim, runAnim, shootAnim, dan dieAnim yang masing-masing menyimpan animasi berbeda sesuai keadaan musuh. Variabel stateTime menyimpan waktu berjalan animasi, sedangkan state dan prevState digunakan untuk menentukan kondisi musuh seperti IDLE, WALK, RUN, SHOOT, atau DIE. Bagian AI menyimpan beberapa variabel tambahan, seperti patrolTargetX yang digunakan untuk menentukan titik tujuan patroli musuh, detectDelay dan delayTimer untuk mengatur jeda sebelum musuh bereaksi terhadap player, detectedPlayer untuk menandai apakah musuh sudah menyadari posisi player, serta damageCooldown untuk mencegah musuh menerima damage terlalu cepat. Variabel countedKill digunakan untuk memastikan musuh hanya dihitung satu kali ketika mati. Terakhir terdapat fireTime dan firedThisShot yang digunakan dalam logika menembak, memastikan bahwa musuh hanya menembakkan peluru satu kali per animasi tembak.

```
public Enemy(float startX, float startY, 1 usage & lfunzz *
            TextureRegion[] idleFrames,
            TextureRegion[] walkFrames,
            TextureRegion[] runFrames,
            TextureRegion[] shootFrames,
            TextureRegion[] dieFrames,
            float scale) {

    this.scale = scale;
    setPosition(startX, startY);

    // Buat animasi
    idleAnim = new Animation<>( frameDuration: 0.2f, idleFrames);
    idleAnim.setPlayMode(Animation.PlayMode.LOOP);
    walkAnim = new Animation<>( frameDuration: 0.08f, walkFrames);
    walkAnim.setPlayMode(Animation.PlayMode.LOOP);
    runAnim = new Animation<>( frameDuration: 0.08f, runFrames);
    runAnim.setPlayMode(Animation.PlayMode.LOOP);
    shootAnim = new Animation<>( frameDuration: 0.08f, shootFrames);
    shootAnim.setPlayMode(Animation.PlayMode.NORMAL);
    dieAnim = new Animation<>( frameDuration: 0.12f, dieFrames);
    dieAnim.setPlayMode(Animation.PlayMode.NORMAL);

    TextureRegion firstFrame = idleFrames[0];
    float w = firstFrame.getRegionWidth() * scale;
    float h = firstFrame.getRegionHeight() * scale;
    setSize(w, h);

    hitbox = new Rectangle( x: getX() + getWidth() * 0.25f,
                           y: getY() + getHeight() * 0.001f,
                           width: getWidth() * 0.5f,
                           height: getHeight() * 0.75f);

    // inisialisasi lastShootTime biar ga langsung menembak
    lastShootTime = TimeUtils.nanoTime();

    pickNewPatrolTarget();
}
```

Selanjutnya adalah constructor untuk Enemy dimana program akan menyimpan posisi awal dari enemy dan juga memecah animasi agar menjadi lebih halus dan menyimpannya pada variabel penyimpan animasi. Kemudian program juga membuat hitbox untuk menyimpan hitbox dari Enemy dan Enemy langsung mencari target terbaru.

```

@Override
public void act(float delta) {
    super.act(delta);
    if (dead) {
        deathTime += delta;
        state = EnemyState.DIE;
        return;
    }

    // update waktu state
    stateTime += delta;

    Player player = findPlayer();
    if (player == null) {
        // tidak ada player di stage: patrol
        state = EnemyState.WALK;
        patrol(delta);
        updateHitbox();
        return;
    }

    // jarak ke player (center)
    float dist = Math.abs(getCenterX() - (player.getHitbox().x + player.getHitbox().width/2f));

    if (dist <= 450f) detectedPlayer = true;

    if (detectedPlayer) {
        delayTimer += delta;
        if (delayTimer < detectDelay) {
            state = EnemyState.IDLE;
        } else {
            // jika dekat cukup untuk menembak -> berhenti di tempat dan mulai animasi SHOOT
            if (dist <= 250f) {
                // Jika dalam radius menembak, berhenti; jika cooldown ready -> SHOOT, else tetap IDLE
                if (canShoot()) {
                    if (state != EnemyState.SHOOT) {
                        stateTime = 0f;
                        firedThisShot = false;
                    }
                    state = EnemyState.SHOOT;
                } else {
                    // cooldown belum selesai -> tetap diam (IDLE)
                    state = EnemyState.IDLE;
                }
            } else if (dist <= 600f) {
                // player agak jauh: kejar
                state = EnemyState.RUN;
                chasePlayer(player, delta);
            } else {
                // player terlihat tapi jauh -> patrol (atau jalan)
                state = EnemyState.WALK;
                patrol(delta);
            }
        }
    } else {
        state = EnemyState.WALK;
        patrol(delta);
    }
}

```

```

        // jika sedang dalam state SHOOT, cek apakah animasi sudah melewati titik fireTime
        if (state == EnemyState.SHOOT) {
            // tidak bergerak saat menembak (patrol/chase berhenti)
            // setelah animasi SHOOT selesai, reset ke IDLE dan set lastShootTime
            if (shootAnim.isAnimationFinished(stateTime)) {
                stateTime = 0f;
                state = EnemyState.IDLE;
                lastShootTime = TimeUtils.nanoTime();
                // setelah selesai, delayTimer tetap (biar tidak langsung reload)
            }
        }

        if (damageCooldown > 0) damageCooldown -= delta;

        updateHitbox();
        prevState = state;
        clampPosition();
    }
}

```

Selanjutnya adalah fungsi act yang digunakan agar Enemy dapat berperilaku sesuai dengan kondisi, semisal Enemy belum menemukan target maka ia akan berpatroli mencari player, apa bila ia sudah menemukan player, maka ia akan mengejarnya, dan jika jangkauannya sudah cukup dekat maka ia akan mulai menembaki player.

```

private Player findPlayer() { 1usage += lfunzz
    if (getStage() == null) return null;
    Actor a = getStage().getRoot().findActor( name: "player");
    return (a instanceof Player) ? (Player) a : null;
}

private float getCenterX() { return getX() + getWidth() / 2f; }

private void clampPosition() { 3 usages += lfunzz
    if (getX() < 0) setX(0);
    if (getX() + getWidth() > 5000) setX(5000 - getWidth());
}

private void chasePlayer(Player player, float delta) { 1usage += lfunzz +1
    float playerCenter = player.getHitbox().x + player.getHitbox().width / 2f;
    if (playerCenter > getCenterX()) {
        setX(getX() + speed * 1.5f * delta); // run lebih cepat
        facingRight = true;
    } else {
        setX(getX() - speed * 1.5f * delta);
        facingRight = false;
    }
    clampPosition();
}

```

Fungsi findPlayer digunakan untuk mengecek apakah player sudah ditemukan. Fungsi getCenter digunakan untuk mengembalikan nilai dari posisi tengah Enemy. Lalu clampPosition digunakan untuk memberi batas pada enemy agar tidak bisa keluar dari kordinat x 0 sampai 5000. Selanjutnya fungsi chasePlayer digunakan agar enemy dapat mengejar player dan mendekati titik x dari player.

```

public boolean shouldFire() { 3usages += !runzz *
    if (state != EnemyState.SHOOT) return false;
    if (firedThisShot) return false;

    if (stateTime >= fireTime) {
        firedThisShot = true;
        return true;
    }
    return false;
}

private void patrol(float delta) { 3usages += !runzz +1
    if (facingRight) {
        setX(getX() + speed * delta);
        if (getX() >= patrolTargetX) pickNewPatrolTarget();
    } else {
        setX(getX() - speed * delta);
        if (getX() <= patrolTargetX) pickNewPatrolTarget();
    }
    clampPosition();
}

private void pickNewPatrolTarget() { 3usages += !runzz
    float move = MathUtils.random(150, 400);
    if (MathUtils.randomBoolean()) {
        patrolTargetX = getX() + move;
        facingRight = true;
    } else {
        patrolTargetX = Math.max(0, getX() - move);
        facingRight = false;
    }
}

public boolean canShoot() { return TimeUtils.nanoTime() - lastShootTime > 1_800_000_000L; // 1.8 detik cooldown }

```

Selanjutnya adalah fungsi shouldFire yang berfungsi untuk mengecek apakah Enemy lebih baik menembak atau tidak. Kemudian fungsi patrol digunakan agar Enemy dapat bergerak ke kanan atau ke kiri ketika melakukan patroli mencari player. Kemudian fungsi pickNewPatrolTarget digunakan agar Enemy dapat bergerak ke posisi random dari 150 sampai 400 + posisi Enemy ketika mencari player. Kemudian fungsi canShoot digunakan untuk mengecek apakah Enemy sudah bisa menembak ketika sudah melewati batas cooldown.

```

public void takeDamage(float dmg) { 1 usage + Asxarthoz +1
    if (dead || damageCooldown > 0) return;
    damageCooldown = 0.3f;
    hp -= dmg;
    if (hp <= 0) {
        dead = true;
        hp = 0;
        stateTime = 0f;
        deathTime = 0f;
    }
}

private void updateHitbox() { 2 usages + Irfunzz
    // hitbox selalu berada relatif ke pos actor - tidak mengambang lagi
    hitbox.setPosition( x: getX() + getWidth() * 0.25f, y: getY() + getHeight() * 0.001f);
}

@Override + Irfunzz
public void draw(Batch batch, float parentAlpha) {
    TextureRegion frame;

    if (state == EnemyState.DIE) {
        frame = dieAnim.getKeyFrame(deathTime, looping: false);
    } else if (state == EnemyState.SHOOT) {
        frame = shootAnim.getKeyFrame(stateTime, looping: false);
    } else if (state == EnemyState.RUN) {
        frame = runAnim.getKeyFrame(stateTime, looping: true);
    } else if (state == EnemyState.WALK) {
        frame = walkAnim.getKeyFrame(stateTime, looping: true);
    } else {
        frame = idleAnim.getKeyFrame(stateTime, looping: true);
    }

    // buat salinan region sehingga flip tidak merusak region sumber
    TextureRegion r = new TextureRegion(frame);
    if (!facingRight && !r.isFlipX()) r.flip( x: true, y: false);
    if (facingRight && r.isFlipX()) r.flip( x: true, y: false);

    // gambar tepat di getX(), getY() - jangan offset, supaya hitbox sinkron
    batch.draw(r, getX(), getY(), getWidth(), getHeight());
}

public Rectangle getHitbox() { return hitbox; } 12 usages + Irfunzz
public boolean isDead() { return dead && dieAnim.isAnimationFinished(deathTime); } 1 usage + Irfunzz

public boolean hasCountedKill() { return countedKill; } 1 usage + Irfunzz
public void setCountedKill(boolean v) { countedKill = v; } 1 usage + Irfunzz

```

Selanjutnya adalah fungsi takeDamage dimana ketika Enemy tidak mati dan damageCooldown = 0 maka ia dapat menerima damage dari player dan nyawanya dikurangi 30. Selanjutnya adalah fungsi draw yang digunakan untuk menggambar Enemy berdasarkan animasi yang sedang dijalankan. Kemudian fungsi getHitbox digunakan untuk mengembalikan variabel hitbox dari Enemy.

Fungsi isDead digunakan untuk mengembalikan nilai apakah Enemy sudah mati atau belum. Fungsi hasCountedKill digunakan untuk mengembalikan nilai dari variabel countedKill. Terakhir fungsi setCountedKill digunakan untuk mengubah nilai dari variabel countedKill.

Player.java

```
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.scenes.scene2d.Actor;
```

Pertama program akan melakukan import untuk Rectangle agar program dapat menjalankan fungsi-fungsi Rectangle, program juga melakukan import untuk Actor agar program dapat melakukan draw, render, dan lain lain.

```
public class Player extends Actor { 11 usages ▾ lfunzz +2

    // Health player
    static final int MAX_HEALTH = 100;  6 usages
    public float health = 100f;  22 usages

    // Physics
    public float x, y;  no usages
    public float speed = 200f;  5 usages
    public float gravity = -800f;  1 usage
    public float velocityY = 0;  4 usages
    public boolean isGround = true;  3 usages

    // Arah & Serangan & sound
    public boolean facingRight = true;  7 usages
    public boolean isAttack = false;  8 usages
    public float attackTime = 0f;  5 usages
    public boolean hitRegistered = false;  6 usages

    Sound hitSound;  2 usages
    Sound dashSound;  2 usages
    Sound healSound;  3 usages
```

Pada class Player, dilakukan extends Actor agar dapat melakukan fungsi-fungsi seperti draw, render, dan lain lain. Player memiliki banyak atribut, pertama terdapat MAX_HEALTH untuk menyimpan batas maksimal nyawa dari player, kemudian health untuk menyimpan data nyawa player. Selanjutnya program juga menyimpan x untuk posisi x player dan y untuk posisi y player, speed digunakan

untuk menyimpan kecepatan dari player dan gravity digunakan untuk menyimpan gaya gravitasi untuk player. VelocityY nantinya digunakan untuk proses lompat pada player dan isGround digunakan untuk mengecek apakah player menginjak dasar atau tidak. Variabel facingRight digunakan untuk menyimpan data apakah player menghadap kanan dan isAttack untuk menyimpan data apakah player sedang melakukan serangan. Variabel attackTime digunakan untuk menyimpan waktu serangan player dan hitRegistered untuk menyimpan data apakah serangan player mengenai musuh. Variabel hitSound digunakan untuk menyimpan suara dari serangan player, dashSound menyimpan suara ketika player melakukan dash, dan healSound digunakan untuk menyimpan suara ketika player melakukan heal.

```
// --- SHIELD VARIABLES ---
public boolean isShielding = false; 10 usages

// --- HEAL VARIABLES ---
public float healTimer = 0f; 6 usages
public float healCooldown = 5f; 1 usage

public Rectangle hitbox; 17 usages
float scale = 0.2f; 6 usages

Animation<TextureRegion> walkAnim; 2 usages
Animation<TextureRegion> attackAnim; 3 usages
TextureRegion[] walkFrames; 6 usages
TextureRegion[] attackFrames; 3 usages
private TextureRegion shieldRegion; 6 usages

// DASH system
boolean isDashing = false; 4 usages
float dashTime = 0f; 3 usages
float dashDuration = 0.15f; 1 usage
float dashSpeed = 900f; 2 usages
float dashCooldown = 3f; 1 usage
float dashCooldownTimer = 0f; 5 usages

float stateTime = 0f; 3 usages
```

Terdapat juga atribut lain seperti isShielding untuk menyimpan data apakah player sedang melakukan shielding, lalu healTimer untuk menyimpan waktu heal dan healCooldown untuk menyimpan cooldown dari heal. Lalu walkAnim untuk menyimpan animasi ketika player berjalan, attackAnim untuk menyimpan animasi

ketika player menyerang, walkFrames untuk menyimpan frame ketika berjalan, attackFrames untuk menyimpan frame ketika player menyerang, dan shieldRegion untuk menyimpan frame ketika player shielding. Kemudian terdapat juga atribut-atribut lain untuk menyimpan sistem dash dari player.

```
// constructor updated
public Player(float x, float y, Texture spriteSheet, Texture attackSheet, Sound sound, Sound soundDash, Sound healSound, Texture
    this.setPosition(x, y);
    this.hitSound = sound;
    this.dashSound = soundDash;
    this.healSound = healSound;
    this.shieldRegion = shieldRegion;

    // character sub-rectangle
    int charX = 750;
    int charY = 410;
    int charW = 585;
    int charH = 920;

    int frameCols = 7;
    int frameRows = 2;
    int frameWidth = spriteSheet.getWidth() / frameCols;
    int frameHeight = spriteSheet.getHeight() / frameRows;
    TextureRegion[][] tmpWalk = TextureRegion.split(spriteSheet, frameWidth, frameHeight);
    TextureRegion[][] tmpAtk = TextureRegion.split(spriteSheet, frameWidth, frameHeight);

    walkFrames = new TextureRegion[7];
    attackFrames = new TextureRegion[7];

    for (int i = 0; i < 7; i++) {
        walkFrames[i] = new TextureRegion(tmpWalk[1][i], charX, charY, charW, charH);
        attackFrames[i] = new TextureRegion(tmpAtk[0][i], charX, charY, charW, charH);
    }

    walkAnim = new Animation<TextureRegion> (frameDuration: 0.1f, walkFrames);
    attackAnim = new Animation<TextureRegion> (frameDuration: 0.1f, attackFrames);

    float w = walkFrames[0].getRegionWidth() * scale;
    float h = walkFrames[0].getRegionHeight() * scale;
    setSize(w, h);
}

hitbox = new Rectangle(getX(), getY(), width: w - 60, height: h - 30);
```

Selanjutnya adalah constructor dari player dimana program akan menyimpan data-data ke atribut dari player dan juga melakukan pemotongan terhadap gambar dari animasi player agar animasi dapat terlihat mulus. Kemudian program juga membuat hitBox agar player dapat menyerang dan menerima serangan.

```
@Override
public void act(float delta) {
    super.act(delta);
    stateTime += delta;

    // Heal Logic
    if (healTimer > 0) {
        healTimer -= delta;
    }

    if (Gdx.input.isKeyJustPressed(Input.Keys.H)) {
        if (healTimer <= 0 && health < MAX_HEALTH) {
            health += 30;
            if (health > MAX_HEALTH) health = MAX_HEALTH;
            healTimer = healCooldown;

            // Play Heal Sound
            if (healSound != null) {
                healSound.play();
            }

            System.out.println("Healed! Current HP: " + health);
        }
    }

    // Shield Logic Input
    if (Gdx.input.isButtonPressed(Input.Buttons.RIGHT)) {
        isShielding = true;
    } else {
        isShielding = false;
    }
}
```

```
// Speed Adjustment Logic
if (isShielding) {
    speed = 100f; // Melambat saat shield
} else {
    // Jika tidak shield, cek sprint
    if (Gdx.input.isKeyPressed(Input.Keys.SHIFT_LEFT)) {
        speed = 550f;
    } else {
        speed = 200f;
    }
}

// Movement
if (Gdx.input.isKeyPressed(Input.Keys.D)) {
    setX(getX() + speed * delta);
    facingRight = true;
}
if (Gdx.input.isKeyPressed(Input.Keys.A)) {
    setX(getX() - speed * delta);
    facingRight = false;
}

// Jump
if (Gdx.input.isKeyPressed(Input.Keys.W) && isGround) {
    velocityY = 400;
    isGround = false;
}

// Dash cooldown
dashCooldownTimer -= delta;
if (Gdx.input.isKeyJustPressed(Input.Keys.SPACE) && dashCooldownTimer <= 0 && !isDashing) {
    isDashing = true;
    this.dashSound.play();
    dashTime = 0f;
    dashCooldownTimer = dashCooldown;
}
```

```
    if (Gdx.input.isButtonJustPressed(Input.Buttons.LEFT) && !isAttack && !isShielding) {
        isAttack = true;
        attackTime = 0f;
        hitRegistered = false;
        hitSound.play();
    }

    // Attack animation logic
    if (isAttack) {
        attackTime += delta;
        if (attackAnim.isAnimationFinished(attackTime)) {
            isAttack = false;
            attackTime = 0f;
        }
    } else {
        hitRegistered = false;
    }

    // Dash execution
    if (isDashing) {
        dashTime += delta;
        if (facingRight) {
            setX(getX() + dashSpeed * delta);
        } else {
            setX(getX() - dashSpeed * delta);
        }
        if (dashTime >= dashDuration){
            isDashing = false;
        }
        hitbox.setPosition(x: getX() + 15, getY());
        return;
    }

    // Gravity
    velocityY += gravity * delta;
    setY(getY() + velocityY * delta);
    if (getY() <= 0) { setY(0); velocityY = 0; isGround = true; }
```

```

// Gravity
velocityY += gravity * delta;
setY(getY() + velocityY * delta);
if (getY() <= 0) { setY(0); velocityY = 0; isGround = true; }

// CLAMP
if (getX() < 0) setX(0);
if (health > MAX_HEALTH) health = MAX_HEALTH;
if (health < 0) health = 0f;

// Update hitbox
hitbox.setPosition(x: getX() + 15, getY());

```

Selanjutnya adalah fungsi act yang digunakan agar player bergerak sesuai keinginan user. Ketika user menekan W maka player akan melompat, A untuk bergerak kiri, D untuk bergerak kanan, spasi untuk dash, klik kiri untuk menyerang, klik kanan untuk shielding, dan H untuk heal. Ketika player berjalan, maka animasi yang ditampilkan adalah animasi player berjalan, ketika menyerang maka animasi yang ditampilkan adalah animasi menyerang, dan jika melakukan shielding maka yang ditampilkan adalah gambar player shielding.

```

public TextureRegion getFrame() { 1 usage ▾ Asxarthoz +1
    TextureRegion rawFrame;
    TextureRegion idle = walkFrames[0];

    if (isAttack) {
        rawFrame = attackAnim.getKeyFrame(attackTime, looping: false);
    } else if (Gdx.input.isKeyPressed(Input.Keys.A) || Gdx.input.isKeyPressed(Input.Keys.D)) {
        rawFrame = walkAnim.getKeyFrame(stateTime, looping: true);
    } else {
        rawFrame = idle;
    }

    TextureRegion frame = new TextureRegion(rawFrame);
    if (facingRight && frame.isFlipX()) frame.flip(x: true, y: false);
    if (!facingRight && !frame.isFlipX()) frame.flip(x: true, y: false);
    return frame;
}

```

Selanjutnya adalah fungsi getFrame untuk mendapatkan frame sesuai dengan input dari player seperti yang sudah dijelaskan sebelumnya.

```

@Override
public void draw(Batch batch, float parentAlpha) {

    if (isShielding) {

        boolean flip = !facingRight;
        if (shieldRegion.isFlipX() != flip) {
            shieldRegion.flip(x: true, y: false);
        }

        float penyesuaianY = -55f;
        float penyesuaianX = -50f;

        batch.draw(
            shieldRegion,
            v: getX() + penyesuaianX,
            v1: getY() + penyesuaianY,
            v2: shieldRegion.getRegionWidth() * scale * 2.2f,
            v3: shieldRegion.getRegionHeight() * scale * 2.2f
        );
    } else {
        TextureRegion frame = getFrame();
        batch.draw(
            frame,
            getX(), getY(),
            v2: frame.getRegionWidth() * scale,
            v3: frame.getRegionHeight() * scale
        );
    }
}

```

Selanjutnya adalah fungsi draw yang berfungsi untuk menggambar frame dari player pada program.

```

public void updateDialog(float delta) { stateTime += delta; }

```

Kemudian updateDialog digunakan untuk menambah stateTime ketika player berbicara dengan NPC.

```
public Rectangle getAttackHitbox() { 2 usages  ± Asxarthoz +1
    if (!isAttack) return null;
    float attackWidth = 60;
    float attackHeight = hitbox.height;
    if (facingRight) {
        return new Rectangle( x: hitbox.x + hitbox.width, hitbox.y, attackWidth, attackHeight);
    } else {
        return new Rectangle( x: hitbox.x - attackWidth, hitbox.y, attackWidth, attackHeight);
    }
}

public Rectangle getHitbox() { return hitbox; } 14 usages  ± lrfunzz

public String getDashStatusString() { 1 usage  ± lrfunzz
    if (dashCooldownTimer <= 0f) return "DASH: READY";
    return String.format("DASH: cooldown %.1fs", dashCooldownTimer);
}
```

Kemudian fungsi getAttackHitbox digunakan untuk mengembalikan variabel hitbox agar enemy atau boss dapat menerima serangan dari player. Lalu getHitbox untuk mengembalikan variabel hitbox. Terakhir getDashStatusString digunakan untuk mengembalikan nilai dari status dash apakah masih cooldown atau tidak.

GameScreen.java

```
package com.gdx;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Camera;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.math.MathUtils;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.utils.Array;
import com.badlogic.gdx.utils.ScreenUtils;
import com.badlogic.gdx.utils.TimeUtils;
import com.badlogic.gdx.utils.viewport.ExtendViewport;
import com.badlogic.gdx.utils.viewport.ScreenViewport;

import java.util.Random;
```

Pada awal file GameScreen ada beberapa library yang dipakai mulai dari library bawaan java seperti `java.util.*;`, sampai library yang telah disediakan oleh libgx seperti `com.badlogic.gdx.*;`

```
 1  public class GameScreen implements Screen {
 2      private final Main game;
 3
 4      private static final int WORLD_WIDTH = 5000;
 5      private static final int WORLD_HEIGHT = 2000;
 6      private static final int V_WIDTH = 800;
 7      private static final int V_HEIGHT = 640;
 8
 9      private int killCount = 0;
10      private static final int KILL_TARGET = 20;
11      private static final int WIN_TARGET = 21;
12      private static final int WAVE_SIZE = 4;
13
14      private Stage gameStage;
15      private Stage uiStage;
16      private ShapeRenderer shapeR;
17      private BitmapFont font;
18
19      private Texture background;
20      private Texture enemyTexture;
21      private Texture playerSheet;
22      private Texture npcSheet;
23      private Texture bulletTexture;
24      private TextureRegion bulletRegion;
25      private Texture shieldTexture;
26      private TextureRegion shieldRegion;
27
28      Sound s_shot, s_hit, s_dash;
29      Sound s_heal;
30      Sound s_die;
31
32      Music bossMusic;
33
34      private Texture healTexture;
35      private Texture gameOverTexture;
36
37      private Player player;
38      private NPC npc;
39      private Array<Enemy> enemies = new Array<>();
40      private Array<Bullet> bullets = new Array<>();
41      private Array<Bullet> bossBullets = new Array<>();
42      private Boss boss;
43      private Array<Bomb> bossBombs = new Array<>();
44      private Array<Explosion> explosions = new Array<>();
45
46      private Texture enemyIdle, enemyWalk, enemyRun, enemyShoot, enemyDie;
47      private TextureRegion[] idleFrames, walkFrames, runFrames, shootFrames, dieFrames;
48
49      private enum GameState {
50          EXPLORE,
51          DIALOG,
52          GAMEOVER,
53          VICTORY
54      }
55      private GameState currentState = GameState.DIALOG;
56      private String[] dialogList = {
57          "Para penjajah berusaha merebut tanah kita.",
58          "Jaga tanah ini dengan sepenuh hatimu!",
59          "Jangan pernah gentar meskipun nyawa taruhannya!"
60      };
61      private int dialogIndex = 0;
62      private float stateTime = 0f;
63      private Random rand = new Random();
64
```

Bagian kode ini berfungsi sebagai deklarasi seluruh variabel yang dibutuhkan oleh `GameScreen`. Semua elemen penting permainan—mulai dari ukuran dunia, objek pemain, musuh, peluru, animasi, suara, hingga state permainan—disiapkan di sini agar dapat digunakan oleh fungsi-fungsi lain dalam kelas.

Kelas `GameScreen` sendiri menggunakan `implements Screen`, sehingga otomatis memperoleh fungsi dasar seperti `show()`, `render()`, `resize()`, `pause()`, dan `dispose()`. Dengan cara ini, `GameScreen` bisa dihubungkan ke sistem layar milik LibGDX dan menjadi salah satu tampilan aktif dalam game.

Di dalam deklarasi ini terdapat beberapa kelompok variabel penting:

- **Konfigurasi dunia dan permainan**, seperti ukuran world, viewport, jumlah kill target, ukuran wave, dan penghitung kill
- **Objek grafis**, meliputi texture, sprite sheet, frame animasi, font, dan renderer
- **Objek audio**, seperti efek tembakan, dash, heal, kematian, serta musik bos
- **Objek gameplay**, mencakup Player, NPC, Boss, serta daftar musuh, peluru, bom, dan ledakan
- **State permainan**, yang mengatur apakah game sedang eksplorasi, dialog, game over, atau victory
- **Dialog awal**, yang muncul sebelum permainan benar-benar dimulai.

Semua deklarasi ini menjadi fondasi dari `GameScreen`, karena setiap variabel digunakan untuk mengatur logika, tampilan, suara, dan interaksi yang akan dijalankan dalam metode-metode berikutnya. Setelah bagian ini siap, barulah gameplay dapat berjalan dengan stabil dan terstruktur.



Bagian ini berguna untuk menginisialisasi game serta load asset agar ketika masuk ke game loading masuknya tidak terlalu lama karena asset nya sudah di load terlebih dahulu.



```
1 private void preloadAsset() {
2     shaperR = new ShapeRenderer();
3     font = new BitmapFont();
4
5     s_shot = Gdx.audio.newSound(Gdx.files.internal("sound/shot.mp3"));
6     s_hit = Gdx.audio.newSound(Gdx.files.internal("sound/hit.mp3"));
7     s_dash = Gdx.audio.newSound(Gdx.files.internal("sound/dash.mp3"));
8     s_heal = Gdx.audio.newSound(Gdx.files.internal("sound/heal.ogg"));
9     s_die = Gdx.audio.newSound(Gdx.files.internal("sound/mati.ogg"));
10
11    bossMusic = Gdx.audio.newMusic(Gdx.files.internal("sound/boss.ogg"));
12    bossMusic.setLooping(true);
13
14    background = new Texture("bg.png");
15    enemyTexture = new Texture("enemy.png");
16    playerSheet = new Texture("sprite.png");
17    npcSheet = new Texture("sprite.png");
18    bulletTexture = new Texture("bullet.png");
19    bulletRegion = new TextureRegion(bulletTexture);
20    shieldTexture = new Texture("shield.png");
21    shieldRegion = new TextureRegion(shieldTexture);
22    healTexture = new Texture("heal.png");
23    gameOverTexture = new Texture("gameover.png");
24
25    enemyIdle = new Texture("shooter/Soldier_1/Idle.png");
26    enemyWalk = new Texture("shooter/Soldier_1/Walk.png");
27    enemyRun = new Texture("shooter/Soldier_1/Run.png");
28    enemyShoot = new Texture("shooter/Soldier_1/Shot_1.png");
29    enemyDie = new Texture("shooter/Soldier_1/Dead.png");
30
31    boss = new Boss(
32        5100, 0,
33        new Texture("Standing.png"),
34        new Texture("Run.png"),
35        new Texture("Roll.png"),
36        new Texture("bullet.png"),
37        4.25f, s_shot
38    );
39}
```

Fungsi `preloadAsset()` bertugas menyiapkan seluruh aset yang dibutuhkan oleh game sebelum gameplay dimulai. Semua variabel yang berkaitan dengan tampilan maupun suara diinisialisasi di sini agar siap dipakai oleh bagian lain dari `GameScreen`.

Fungsi ini memuat beberapa kelompok aset:

1. **Objek dasar untuk rendering**

`ShapeRenderer` dan `BitmapFont` dibuat untuk menggambar elemen visual seperti health bar dan teks UI.

2. **Sound effect dan musik**

Berbagai efek suara—tembakan, kena hit, dash, heal, dan kematian—dimuat

menggunakan `Gdx.audio.newSound()`. Musik bos dimuat sebagai `Music`, diatur agar looping, dan nanti diputar saat pertarungan bos dimulai.

3. Texture utama

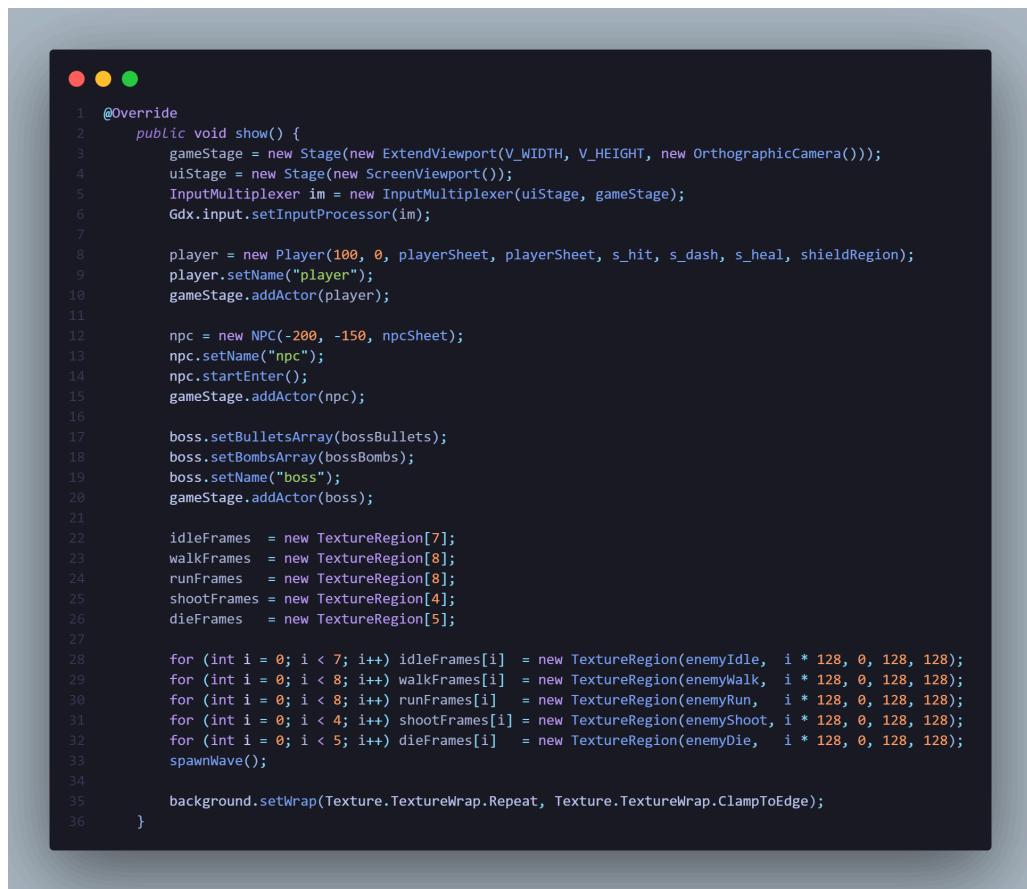
Semua gambar yang diperlukan seperti background, karakter pemain, NPC, musuh, bullet, shield, ikon heal, dan tampilan game over dimuat sebagai `Texture` atau `TextureRegion`

4. Sprite musuh dan animasi

Texture khusus untuk animasi musuh (idle, jalan, lari, menembak, dan mati) dimuat agar bisa dipotong menjadi frame animasi.

5. Inisialisasi Boss

Objek `Boss` dibuat lengkap dengan posisi awal, tekstur animasi, tekstur roll, bullet, ukuran skala, dan efek suara tembakan.



```
1  @Override
2      public void show() {
3          gameStage = new Stage(new ExtendViewport(V_WIDTH, V_HEIGHT, new OrthographicCamera()));
4          uiStage = new Stage(new ScreenViewport());
5          InputMultiplexer im = new InputMultiplexer(uiStage, gameStage);
6          Gdx.input.setInputProcessor(im);
7
8          player = new Player(100, 0, playerSheet, playerSheet, s_hit, s_dash, s_heal, shieldRegion);
9          player.setName("player");
10         gameStage.addActor(player);
11
12         npc = new NPC(-200, -150, npcSheet);
13         npc.setName("npc");
14         npc.startEnter();
15         gameStage.addActor(npc);
16
17         boss.setBulletsArray(bossBullets);
18         boss.setBombsArray(bossBombs);
19         boss.setName("boss");
20         gameStage.addActor(boss);
21
22         idleFrames = new TextureRegion[7];
23         walkFrames = new TextureRegion[8];
24         runFrames = new TextureRegion[8];
25         shootFrames = new TextureRegion[4];
26         dieFrames = new TextureRegion[5];
27
28         for (int i = 0; i < 7; i++) idleFrames[i] = new TextureRegion(enemyIdle, i * 128, 0, 128, 128);
29         for (int i = 0; i < 8; i++) walkFrames[i] = new TextureRegion(enemyWalk, i * 128, 0, 128, 128);
30         for (int i = 0; i < 8; i++) runFrames[i] = new TextureRegion(enemyRun, i * 128, 0, 128, 128);
31         for (int i = 0; i < 4; i++) shootFrames[i] = new TextureRegion(enemyShoot, i * 128, 0, 128, 128);
32         for (int i = 0; i < 5; i++) dieFrames[i] = new TextureRegion(enemyDie, i * 128, 0, 128, 128);
33         spawnWave();
34
35         background.setWrap(Texture.TextureWrap.Repeat, Texture.TextureWrap.ClampToEdge);
36     }
```

Fungsi `show()` adalah salah satu metode inti dari interface `Screen`. Fungsi ini dipanggil sekali ketika `GameScreen` pertama kali ditampilkan. Tujuannya adalah menyiapkan seluruh elemen gameplay yang membutuhkan konteks stage dan input—hal-hal yang tidak dapat dilakukan hanya dengan memuat asset.

Di dalam fungsi ini dilakukan beberapa proses penting:

1. Inisialisasi Stage dan sistem input

Dua stage dibuat: `gameStage` untuk objek game dan `uiStage` untuk elemen antarmuka. `InputMultiplexer` dipasang agar input dapat diterima oleh UI maupun objek game sekaligus.

2. Pembuatan objek gameplay utama

Pemain, NPC, dan bos dibuat dan langsung ditambahkan ke `gameStage`. Di tahap ini objek-objek tersebut sudah siap menjalankan logika dan animasinya masing-masing.

3. Menyiapkan animasi musuh

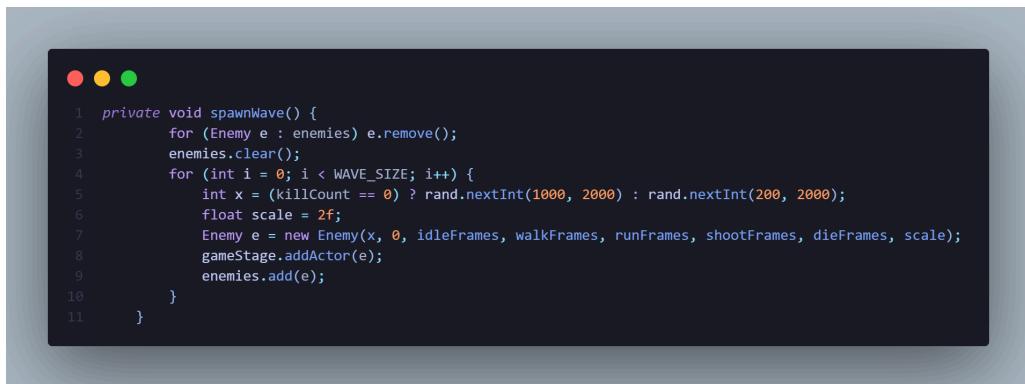
Frame-frame animasi musuh dipotong dari sprite sheet berdasarkan ukuran tiap frame. Proses ini menghasilkan array `idleFrames`, `walkFrames`, `runFrames`, dan seterusnya, yang nanti digunakan untuk animasi hidup musuh.

4. Memulai wave awal

Fungsi `spawnWave()` dipanggil untuk memunculkan musuh pertama sebelum permainan dimulai penuh.

5. Pengaturan teknis texture dan viewport

Texture background dibuat agar bisa diulang secara horizontal, dan viewport Stage memastikan tampilan dunia mengikuti setting ukuran virtual.



```
1 private void spawnWave() {
2     for (Enemy e : enemies) e.remove();
3     enemies.clear();
4     for (int i = 0; i < WAVE_SIZE; i++) {
5         int x = (killCount == 0) ? rand.nextInt(1000, 2000) : rand.nextInt(200, 2000);
6         float scale = 2f;
7         Enemy e = new Enemy(x, 0, idleFrames, walkFrames, runFrames, shootFrames, dieFrames, scale);
8         gameStage.addActor(e);
9         enemies.add(e);
10    }
11 }
```

Fungsi `spawnWave()` ini bertugas membuat satu gelombang musuh baru. Ketika dipanggil, fungsi ini terlebih dahulu menghapus semua musuh yang masih tersisa agar wave baru dimulai dalam keadaan bersih.

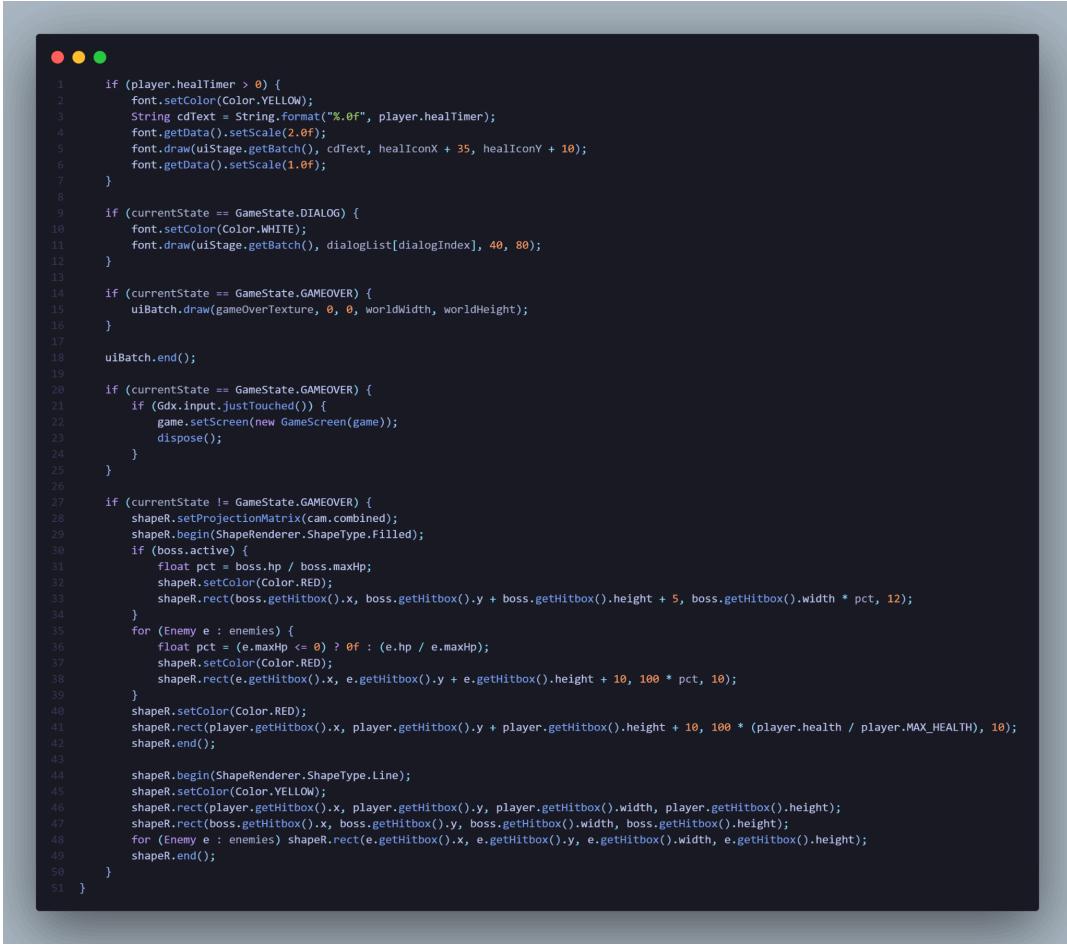
Setiap gelombang akan menghasilkan sejumlah musuh sesuai nilai `WAVE_SIZE` (dalam kasus ini: 4 musuh). Untuk setiap musuh, fungsi ini menentukan posisi spawn secara acak pada sumbu X, sehingga kemunculan musuh tidak selalu di tempat yang sama. Pemilihan area spawn juga sedikit berbeda saat kill count masih nol, sehingga wave pertama memiliki posisi spawn yang lebih jauh.

Setelah posisi acak ditentukan, fungsi membuat objek `Enemy` baru dengan animasi dan skala tertentu, lalu menambahkannya ke `gameStage` serta daftar `enemies`.

```
1  @Override
2  public void render(float d) {
3      ScreenUtils.clear(0, 0, 0, 1);
4      float delta = Gdx.graphics.getDeltaTime();
5      stateTime += delta;
6
7      if (player.health <= 0 && currentState != GameState.GAMEOVER) {
8          currentState = GameState.GAMEOVER;
9          s_die.play();
10         if (bossMusic.isPlaying()) bossMusic.stop();
11     }
12
13     if (currentState != GameState.GAMEOVER) {
14
15         if (currentState == GameState.DIALOG) {
16             player.updateDialog(delta);
17         }
18         gameStage.act(delta);
19         uiStage.act(delta);
20
21         for (Enemy e : enemies) {
22             if (e.shouldFire()) {
23                 shootFromEnemy(e);
24                 e.lastShootTime = TimeUtils.nanoTime();
25             }
26         }
27
28         if (currentState == GameState.DIALOG) {
29             if (Gdx.input.isButtonJustPressed(Input.Buttons.LEFT)) {
30                 dialogIndex++;
31                 if (dialogIndex >= dialogList.length) {
32                     npc.startLeave();
33                     currentState = GameState.EXPLORE;
34                 }
35             }
36         }
37
38         if (player.isAttack) {
39             Rectangle atk = player.getAttackHitbox();
40             if (atk != null && !player.hitRegistered) {
41                 for (Enemy e : enemies) {
42                     if (!e.dead && atk.overlaps(e.getHitbox())) {
43                         e.takeDamage(100);
44                         player.hitRegistered = true;
45                         break;
46                     }
47                 }
48             }
49         }
50
51         if (player.isAttack && boss.active) {
52             Rectangle atk = player.getAttackHitbox();
53             if (atk != null && atk.overlaps(boss.hitbox) && !player.hitRegistered) {
54                 boss.takeDamage(30);
55                 player.hitRegistered = true;
56             }
57         }
58
59     }
```

```
 1  for (int i = enemies.size - 1; i >= 0; i--) {
 2      Enemy en = enemies.get(i);
 3      if (en.isDead()) {
 4          if (!en.hasCountedKill()) {
 5              en.setCountedKill(true);
 6              killCount++;
 7
 8              if (killCount >= KILL_TARGET && !boss.active) {
 9                  boss.setActive(true);
10                  boss.state = Boss.State.RUN;
11                  bossMusic.play();
12              }
13          }
14          en.remove();
15          enemies.removeIndex(i);
16      }
17  }
18
19
20  if (boss.active && boss.hp <= 0) {
21      boss.remove();
22      boss.active = false;
23      killCount = WIN_TARGET;
24      currentState = GameState.VICTORY;
25      if (bossMusic.isPlaying()) bossMusic.stop();
26      game.setScreen(new EndScreen(game));
27      dispose();
28      return;
29  }
30
31  if (enemies.size == 0 && killCount < KILL_TARGET && currentState == GameState.EXPLORE) {
32      spawnWave();
33  }
34
35  for (int i = bullets.size - 1; i >= 0; i--) {
36      Bullet b = bullets.get(i);
37      b.rect.x += b.velX * delta;
38      b.rect.y += b.velY * delta;
39
40      if (b.rect.overlaps(player.hitbox)) {
41          if (player.isShielding) {
42              bullets.removeIndex(i);
43              continue;
44          }
45
46          if (player.health > 0) player.health -= 10;
47          bullets.removeIndex(i);
48          continue;
49      }
50
51      if (b.rect.x < 0 || b.rect.x > WORLD_WIDTH || b.rect.y < 0 || b.rect.y > WORLD_HEIGHT) {
52          bullets.removeIndex(i);
53      }
54  }
55  for (int i = bossBombs.size - 1; i >= 0; i--) {
56      Bomb b = bossBombs.get(i);
57      b.update(delta);
58
59      if (b.rect.overlaps(player.hitbox) && player.isShielding) {
60          explosions.add(new Explosion(b.rect.x + 25, b.rect.y + 25, boss.getExplosionAnim()));
61          bossBombs.removeIndex(i);
62          continue;
63      }
64      if (b.rect.overlaps(player.hitbox) && !player.isShielding) {
65          player.health -= 20;
66          if (player.health < 0) player.health = 0;
67      }
68      if (b.exploded) {
69          explosions.add(new Explosion(b.rect.x + 25, b.rect.y + 25, boss.getExplosionAnim()));
70          s_hit.play();
71          bossBombs.removeIndex(i);
72      }
73  }
}
```

```
1  for (int i = explosions.size - 1; i >= 0; i--) {
2      Explosion ex = explosions.get(i);
3      ex.update(delta);
4
5      if (ex.life <= 0) {
6          explosions.removeIndex(i);
7          continue;
8      }
9      float dx = (player.hitbox.x + player.hitbox.width/2) - ex.x;
10     float dy = (player.hitbox.y + player.hitbox.height/2) - ex.y;
11     float dist = (float)Math.sqrt(dx*dx + dy*dy);
12
13     if (!ex.hasDamaged && dist <= ex.radius) {
14         if (!player.isShielding && player.health > 0) {
15             player.health -= 40;
16         }
17         ex.hasDamaged = true;
18     }
19     if (ex.life <= 0) explosions.removeIndex(i);
20 }
21
22 Camera cam = gameStage.getCamera();
23 float halfW = ((ExtendViewport)gameStage.getViewport()).getWorldWidth() / 2f;
24 float px = player.getX() + player.getWidth() / 2f;
25 cam.position.x = MathUtils.clamp(px, halfW, WORLD_WIDTH - halfW);
26 cam.update();
27
28 SpriteBatch batch = (SpriteBatch)gameStage.getBatch();
29 batch.setProjectionMatrix(cam.combined);
30 batch.begin();
31 batch.draw(background, 0, 0);
32 batch.draw(background, background.getWidth(), 0);
33 batch.draw(background, 2 * background.getWidth(), 0);
34 batch.end();
35
36 gameStage.draw();
37
38 batch.begin();
39 for (Bomb b : bossBombs) {
40     if (!b.exploded) batch.draw(b.sprite, b.rect.x, b.rect.y, 50, 50);
41 }
42 for (int i = explosions.size - 1; i >= 0; i--) {
43     Explosion ex = explosions.get(i);
44     ex.update(delta);
45     if (ex.life <= 0) {
46         explosions.removeIndex(i);
47         continue;
48     }
49     TextureRegion frame = ex.getFrame();
50     batch.draw(frame, ex.x - 64, ex.y - 64, 128, 128);
51 }
52 batch.end();
53
54 SpriteBatch uiBatch = (SpriteBatch)uiStage.getBatch();
55 uiBatch.setProjectionMatrix(uiStage.getCamera().combined);
56
57 uiBatch.begin();
58 float worldWidth = uiStage.getViewport().getWorldWidth();
59 float worldHeight = uiStage.getViewport().getWorldHeight();
60
61 font.setColor(Color.WHITE);
62 font.draw(uiStage.getBatch(), "Kills: " + killCount + " / " + KILL_TARGET, 20, worldHeight - 20);
63 font.draw(uiStage.getBatch(), player.getDashStatusString(), 20, worldHeight - 40);
64
65 float healIconX = 150;
66 float healIconY = worldHeight - 90;
67 float healIconSize = 90;
68
```



```
1     if (player.healTimer > 0) {
2         font.setColor(Color.YELLOW);
3         String cdText = String.format("%.0f", player.healTimer);
4         font.getData().setScale(2.0f);
5         font.draw(uiStage.getBatch(), cdText, healIconX + 35, healIconY + 10);
6         font.getData().setScale(1.0f);
7     }
8
9     if (currentState == GameState.DIALOG) {
10        font.setColor(Color.WHITE);
11        font.draw(uiStage.getBatch(), dialogList[dialogIndex], 40, 80);
12    }
13
14     if (currentState == GameState.GAMEOVER) {
15        uiBatch.draw(gameOverTexture, 0, 0, worldWidth, worldHeight);
16    }
17
18    uiBatch.end();
19
20    if (currentState == GameState.GAMEOVER) {
21        if (Gdx.input.justTouched()) {
22            game.setScreen(new GameScreen(game));
23            dispose();
24        }
25    }
26
27    if (currentState != GameState.GAMEOVER) {
28        shapeR.setProjectionMatrix(cam.combined);
29        shapeR.begin(ShapeRenderer.ShapeType.Filled);
30        if (boss.active) {
31            float pct = boss.hp / boss.maxHp;
32            shapeR.setColor(Color.RED);
33            shapeR.rect(boss.getHitbox().x, boss.getHitbox().y + boss.getHitbox().height + 5, boss.getHitbox().width * pct, 12);
34        }
35        for (Enemy e : enemies) {
36            float pct = (e.maxHp <= 0) ? 0f : (e.hp / e.maxHp);
37            shapeR.setColor(Color.RED);
38            shapeR.rect(e.getHitbox().x, e.getHitbox().y + e.getHitbox().height + 10, 100 * pct, 10);
39        }
40        shapeR.setColor(Color.RED);
41        shapeR.rect(player.getHitbox().x, player.getHitbox().y + player.getHitbox().height + 10, 100 * (player.health / player.MAX_HEALTH), 10);
42        shapeR.end();
43
44        shapeR.begin(ShapeRenderer.ShapeType.Line);
45        shapeR.setColor(Color.YELLOW);
46        shapeR.rect(player.getHitbox().x, player.getHitbox().y, player.getHitbox().width, player.getHitbox().height);
47        shapeR.rect(boss.getHitbox().x, boss.getHitbox().y, boss.getHitbox().width, boss.getHitbox().height);
48        for (Enemy e : enemies) shapeR.rect(e.getHitbox().x, e.getHitbox().y, e.getHitbox().width, e.getHitbox().height);
49        shapeR.end();
50    }
51 }
```

Fungsi `render()` ini adalah inti dari proses pembaruan dan penampilan game setiap frame. Setiap kali layar diperbarui, fungsi inilah yang menangani logika permainan, memeriksa kondisi game, serta menggambar seluruh elemen visual.

Secara garis besar, fungsi ini melakukan empat tugas utama:

1. Memperbarui logika permainan setiap frame.

Fungsi ini membaca input pemain, menghitung waktu (`delta time`), lalu memperbarui state game seperti dialog, pergerakan stage, aksi musuh, serta kondisi pemain. Contohnya, ketika pemain menyerang, fungsi ini mengecek apakah serangan mengenai musuh, atau ketika musuh menembak, fungsi ini memutuskan kapan peluru dikeluarkan.

2. Mengatur transisi state game.

Beberapa kondisi penting diproses di sini:

- Ketika nyawa pemain habis, state berubah menjadi *GAMEOVER*
- Ketika jumlah musuh terbunuh mencapai target, bos akan diaktifkan
- Ketika bos mati, state berubah menjadi *VICTORY* dan game berpindah ke layar akhir.

Fungsi ini memastikan alur permainan berjalan sesuai desain.

3. Mengelola objek dinamis seperti musuh, peluru, bom, dan ledakan.

Setiap objek yang bergerak atau dapat hilang diperbarui dan dicek kondisinya. Peluru bergerak berdasarkan velocity, bom bos meledak setelah waktunya, dan ledakan memberikan damage area. Semua pengecekan hitbox dan penghapusan objek yang sudah habis dilakukan di sini.

4. Menggambar seluruh elemen visual.

Mulai dari background, karakter, musuh, bom, animasi ledakan, hingga UI seperti jumlah kill, cooldown heal, dialog, dan tampilan *GAMEOVER*. Termasuk juga menggambar health bar pemain, musuh, dan bos menggunakan ShapeRenderer.



```
1 void shootFromEnemy(Enemy e) {
2     float startX = e.getHitbox().x + e.getHitbox().width / 2;
3     if (e.facingRight) startX += 30; else startX -= 130;
4     float startY = e.getHitbox().y + e.getHitbox().height / 2 - 25;
5
6     float targetX = player.getHitbox().x + player.getHitbox().width / 2;
7     float bulletSpeed = 500f;
8     float velX = (targetX > startX) ? bulletSpeed : -bulletSpeed;
9     float velY = 0;
10
11    Bullet bullet = new Bullet(bulletRegion, startX, startY, 20, 20, velX, velY, player);
12    bullet.rotation = (velX < 0) ? 0 : 180;
13    bullets.add(bullet);
14    s_shot.play();
15    gameStage.addActor(bullet);
16
17    e.lastShootTime = TimeUtils.nanoTime();
18}
```

Fungsi `shootFromEnemy()` dipakai agar musuh bisa menembak ke arah pemain. Fungsi ini menentukan posisi awal peluru berdasarkan arah musuh, menghitung apakah peluru harus bergerak ke kiri atau kanan, lalu membuat objek peluru dengan kecepatan yang sesuai. Peluru dimasukkan ke `gameStage`, ditambahkan ke daftar peluru, dan efek suara tembakan diputar. Terakhir, waktu tembak musuh diperbarui supaya tidak menembak terlalu cepat.

```
1  @Override  
2      public void resize(int width, int height) {  
3          gameStage.setViewport().update(width, height, true);  
4          uiStage.setViewport().update(width, height, true);  
5      }
```

Fungsi `resize()` digunakan untuk menyesuaikan tampilan ketika ukuran layar berubah. Di sini, viewport milik `gameStage` dan `uiStage` diperbarui agar semua elemen tetap tampil dengan skala dan posisi yang benar meskipun jendela game diperbesar atau diperkecil.

```
1  @Override public void hide() {}  
2  @Override public void pause() {}  
3  @Override public void resume() {}
```

Fungsi-fungsi ini adalah bagian dari interface `Screen`, namun pada `GameScreen` tidak digunakan sehingga dibiarkan kosong.

- `hide()` dipanggil ketika layar ini diganti layar lain
- `pause()` dipanggil saat game dijeda
- `resume()` dipanggil saat game dilanjutkan kembali.

Karena game tidak membutuhkan perilaku khusus di saat-saat tersebut, fungsi-fungsi ini cukup dibiarkan tanpa isi.

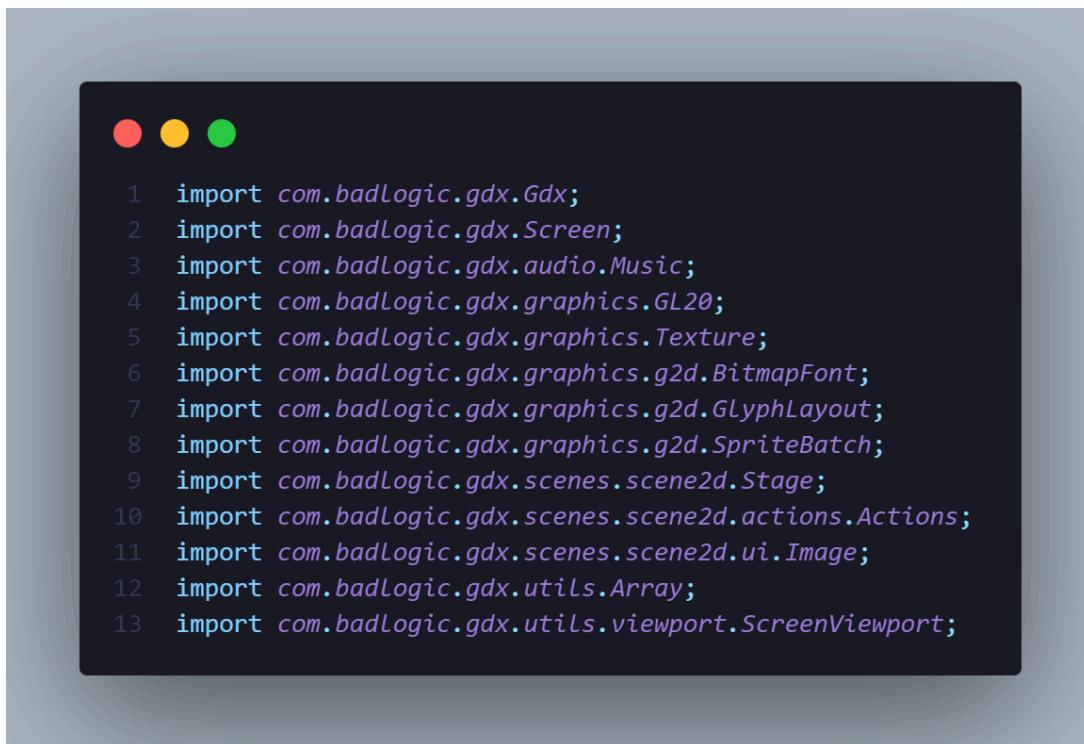
```
1  @Override
2      public void dispose() {
3          shapeR.dispose();
4          font.dispose();
5          gameStage.dispose();
6          uiStage.dispose();
7          background.dispose();
8          enemyTexture.dispose();
9          playerSheet.dispose();
10         npcSheet.dispose();
11         bulletTexture.dispose();
12         shieldTexture.dispose();
13         healTexture.dispose();
14         gameOverTexture.dispose();
15
16         s_shot.dispose();
17         s_hit.dispose();
18         s_dash.dispose();
19         s_heal.dispose();
20         s_die.dispose();
21
22         bossMusic.dispose();
23     }
```

Fungsi `dispose()` digunakan untuk membersihkan seluruh resource yang dipakai game agar tidak terjadi memory leak.

Semua objek seperti `Stage`, `Texture`, `Sound`, `Music`, `BitmapFont`, dan `ShapeRenderer` disimpan di GPU atau memori native LibGDX. Karena itu, semuanya harus dipanggil `dispose()` ketika `GameScreen` tidak digunakan lagi.

Dengan kata lain, fungsi ini memastikan game tetap efisien dan stabil dengan melepaskan seluruh aset yang sudah tidak diperlukan.

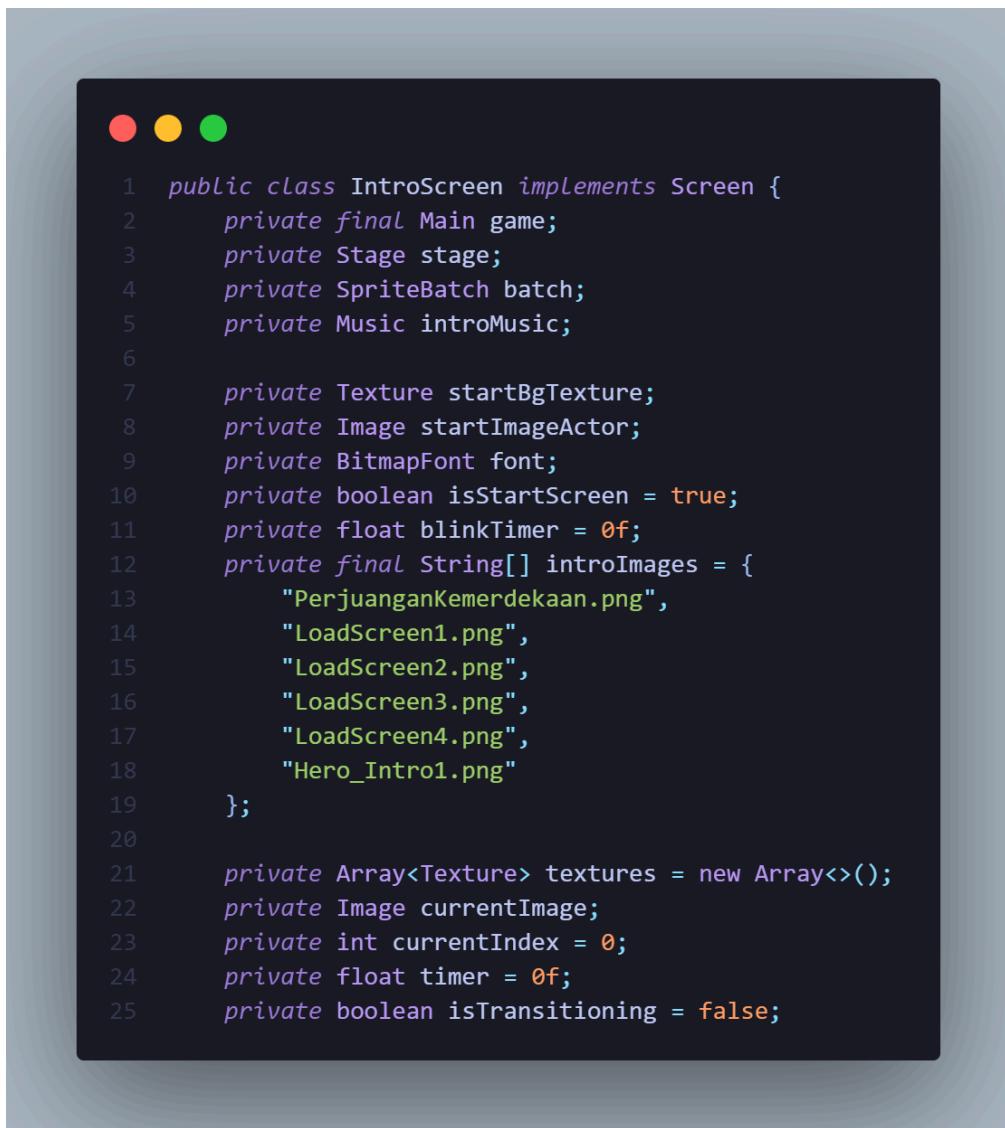
IntroScreen.java



Bagian ini berisi daftar *import* yang digunakan oleh GameScreen. Setiap baris membawa kelas dari LibGDX agar bisa dipakai tanpa menulis nama lengkap paketnya.

Secara sederhana, ini adalah daftar “peralatan” yang dipakai dalam file: mulai dari kelas untuk grafik (Texture, SpriteBatch, BitmapFont), audio (Music), manajemen stage (Stage, Image, Actions), hingga utilitas seperti Array dan ScreenViewport.

Tidak ada logika khusus di bagian ini—tujuannya hanya memastikan semua kelas yang diperlukan tersedia saat file dikompilasi.



```
1 public class IntroScreen implements Screen {
2     private final Main game;
3     private Stage stage;
4     private SpriteBatch batch;
5     private Music introMusic;
6
7     private Texture startBgTexture;
8     private Image startImageActor;
9     private BitmapFont font;
10    private boolean isStartScreen = true;
11    private float blinkTimer = 0f;
12    private final String[] introImages = {
13        "PerjuanganKemerdekaan.png",
14        "LoadScreen1.png",
15        "LoadScreen2.png",
16        "LoadScreen3.png",
17        "LoadScreen4.png",
18        "Hero_Intro1.png"
19    };
20
21    private Array<Texture> textures = new Array<>();
22    private Image currentImage;
23    private int currentIndex = 0;
24    private float timer = 0f;
25    private boolean isTransitioning = false;
```

Bagian ini adalah deklarasi variabel untuk `IntroScreen`, yaitu layar pembuka sebelum game dimulai. Semua elemen yang dibutuhkan intro disiapkan di sini: game utama, stage, batch render, musik intro, texture background, font, serta flag untuk mengatur apakah layar masih berada di fase “start”.

Array `introImages` menampung daftar nama file gambar yang akan ditampilkan secara berurutan. Selain itu disiapkan juga array `textures`, variabel `currentImage`, index gambar aktif, timer pergantian gambar, dan flag `isTransitioning` untuk mengatur proses perpindahan antar-gambar.

Secara keseluruhan, bagian ini memuat semua data dasar yang dibutuhkan untuk menjalankan animasi intro dan transisinya sebelum masuk ke gameplay.



```
1 public IntroScreen(Main game) {
2     this.game = game;
3     stage = new Stage(new ScreenViewport());
4     batch = new SpriteBatch();
5     font = new BitmapFont();
6     font.getData().setScale(2f);
7     try {
8         introMusic = Gdx.audio.newMusic(Gdx.files.internal("sound/soundEpic.ogg"));
9         introMusic.setLooping(true);
10        introMusic.setVolume(0.5f);
11        introMusic.play();
12    } catch (Exception e) {
13        System.out.println("Error memuat musik: " + e.getMessage());
14    }
15    startBgTexture = new Texture("start.png");
16    startImageActor = new Image(startBgTexture);
17    startImageActor.setFillParent(true);
18    stage.addActor(startImageActor);
19 }
```

Konstruktor `IntroScreen` ini bertugas menyiapkan seluruh elemen awal dari layar intro. Di dalamnya dibuat `stage`, `SpriteBatch`, dan `BitmapFont` sebagai dasar untuk menampilkan gambar dan teks. Musik intro juga dimuat melalui `Gdx.audio.newMusic()`, kemudian disetel agar berjalan berulang, diberi volume yang sesuai, dan langsung diputar, dengan penanganan error menggunakan blok `try-catch`. Setelah itu, gambar awal intro (`start.png`) dimuat sebagai `Texture`, diubah menjadi `Image`, diatur agar memenuhi layar, dan ditambahkan ke `stage`. Dengan langkah-langkah ini, layar intro sudah siap tampil lengkap dengan musik dan gambar pembuka sebelum transisi intro berikutnya dijalankan.



```
1 private void startIntroSequence() {
2     isStartScreen = false;
3     startImageActor.addAction(Actions.sequence(
4         Actions.fadeOut(0.5f),
5         Actions.run(() -> {
6             startImageActor.remove();
7             loadNextImage();
8         })
9     ));
10 }
```

Fungsi `startIntroSequence()` ini memulai transisi dari layar start menuju rangkaian gambar intro. Begitu dipanggil, status `isStartScreen` diubah menjadi

false untuk menandai bahwa fase layar awal sudah selesai. Setelah itu, gambar start diberi animasi fade-out selama 0.5 detik. Ketika animasi berakhir, gambar tersebut dihapus dari stage, lalu fungsi `loadNextImage()` dipanggil untuk mulai menampilkan gambar intro pertama. Dengan begitu, transisi dari layar start ke intro berjalan halus dan otomatis.



```
1 private void loadNextImage() {
2     if (currentIndex >= introImages.length) {
3         fadeOutAndStartGame();
4         return;
5     }
6     Texture tex = new Texture(introImages[currentIndex]);
7     textures.add(tex);
8     currentImage = new Image(tex);
9     currentImage.setFillParent(true);
10    currentImage.getColor().a = 0f;
11    currentImage.addAction(Actions.fadeIn(1.2f));
12    stage.clear();
13    stage.addActor(currentImage);
14    timer = 0f;
15 }
```

Fungsi `loadNextImage()` ini bertugas menampilkan gambar intro berurutan. Jika indeks gambar sudah melewati jumlah gambar yang tersedia, fungsi langsung memanggil `fadeOutAndStartGame()` untuk mengakhiri intro dan masuk ke game. Jika belum, gambar baru dimuat menjadi `Texture`, disimpan ke daftar `textures`, lalu dibuat menjadi `Image` yang memenuhi layar. Transparansi awal gambar dibuat 0 agar bisa diberi efek fade-in selama 1.2 detik. Setelah itu, stage dibersihkan dan gambar baru ditambahkan sebagai aktor utama. Terakhir, timer di-reset agar sistem pergantian gambar tetap terkontrol. Dengan cara ini, setiap gambar intro ditampilkan satu per satu dengan transisi halus.



```
1  @Override
2  public void render(float delta) {
3      Gdx.gl.glClearColor(0, 0, 0, 1);
4      Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
5
6      stage.act(delta);
7      stage.draw();
8
9      if (isStartScreen) {
10         blinkTimer += delta;
11         float alpha = (float) Math.abs(Math.sin(blinkTimer * 2));
12
13         batch.begin();
14         font.setColor(1, 1, 1, alpha);
15         GlyphLayout layout = new GlyphLayout(font, "TAP ANYWHERE TO START");
16         float fontX = (Gdx.graphics.getWidth() - layout.width) / 2;
17         float fontY = (Gdx.graphics.getHeight() / 2) - 100;
18         font.draw(batch, layout, fontX, fontY);
19         batch.end();
20
21         if (Gdx.input.justTouched()) {
22             startIntroSequence();
23         }
24         return;
25     }
26     if (isTransitioning) return;
27
28     timer += delta;
29     if (timer > 5.5f) {
30         nextImage();
31     }
32
33     if (Gdx.input.justTouched() && !isTransitioning) {
34         nextImage();
35     }
36 }
```

Fungsi `render()` pada `IntroScreen` ini mengatur seluruh alur tampilan intro setiap frame. Pertama, layar dibersihkan dan stage diperbarui serta digambar. Jika masih berada di layar start, fungsi menampilkan teks “TAP ANYWHERE TO START” dengan efek berkedip menggunakan perhitungan sinus. Begitu layar disentuh, intro memasuki tahap transisi dengan memanggil `startIntroSequence()`. Setelah keluar dari layar start, fungsi mengecek apakah sedang dalam transisi; jika ya, proses ditunda sampai transisi selesai. Jika tidak, timer berjalan untuk menentukan kapan gambar intro berikutnya harus dimunculkan, yaitu setiap 5.5 detik. Pengguna juga bisa menyentuh layar untuk maju lebih cepat selama tidak sedang transisi. Dengan logika ini, render mengontrol animasi fade, pergantian gambar, dan respons input selama intro berlangsung.



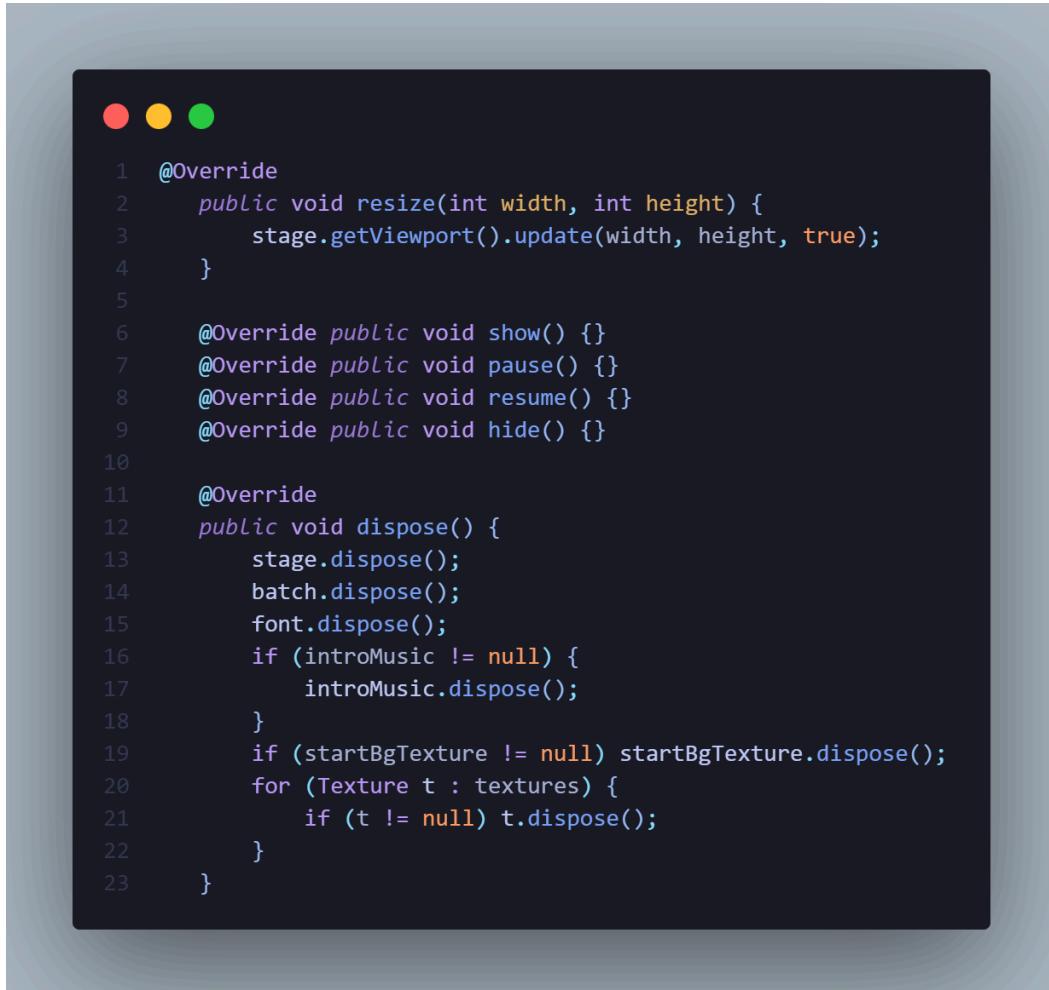
```
1 private void nextImage() {
2     if (isTransitioning) return;
3     isTransitioning = true;
4
5     currentImage.addAction(Actions.sequence(
6         Actions.fadeOut(1.2f),
7         Actions.run(() -> {
8             currentIndex++;
9             isTransitioning = false;
10            loadNextImage();
11        })
12    ));
13 }
```

Fungsi `nextImage()` ini mengatur proses perpindahan dari gambar intro saat ini ke gambar berikutnya. Jika sedang berada dalam transisi, fungsi langsung berhenti untuk mencegah perpindahan ganda. Jika tidak, status `isTransitioning` diaktifkan, lalu gambar yang sedang tampil diberi animasi fade-out selama 1.2 detik. Setelah animasi selesai, indeks gambar dinaikkan, transisi dinonaktifkan, dan fungsi `loadNextImage()` dipanggil untuk memunculkan gambar berikutnya. Mekanisme ini membuat perpindahan setiap slide intro berlangsung halus dan terkontrol.



```
1 private void fadeOutAndStartGame() {
2     isTransitioning = true;
3
4     Runnable switchScreen = () -> {
5         if (introMusic != null && introMusic.isPlaying()) {
6             introMusic.stop();
7         }
8
9         game.setScreen(new GameScreen(game));
10        dispose();
11    };
12
13    if (currentImage != null) {
14        currentImage.addAction(Actions.sequence(
15            Actions.fadeOut(0.4f),
16            Actions.run(switchScreen)
17        ));
18    } else {
19        switchScreen.run();
20    }
21}
```

Fungsi `fadeOutAndStartGame()` adalah langkah terakhir dari intro sebelum game utama dimulai. Fungsi ini mengaktifkan status transisi lalu menyiapkan sebuah `Runnable` bernama `switchScreen`, yang bertugas menghentikan musik intro jika masih berjalan, mengganti layar menjadi `GameScreen`, dan membuang semua resource intro melalui `dispose()`. Jika masih ada gambar intro yang sedang tampil, gambar tersebut diberi animasi fade-out selama 0.4 detik, kemudian `switchScreen` dijalankan setelah animasi selesai. Jika tidak ada gambar aktif, `switchScreen` dijalankan langsung. Dengan cara ini, perpindahan dari intro menuju gameplay berlangsung halus dan bebas gangguan.



```
1 @Override
2     public void resize(int width, int height) {
3         stage.setViewport().update(width, height, true);
4     }
5
6     @Override public void show() {}
7     @Override public void pause() {}
8     @Override public void resume() {}
9     @Override public void hide() {}
10
11    @Override
12    public void dispose() {
13        stage.dispose();
14        batch.dispose();
15        font.dispose();
16        if (introMusic != null) {
17            introMusic.dispose();
18        }
19        if (startBgTexture != null) startBgTexture.dispose();
20        for (Texture t : textures) {
21            if (t != null) t.dispose();
22        }
23    }
```

Bagian akhir ini berisi implementasi beberapa metode dasar `Screen` serta fungsi `dispose()` untuk membersihkan resource. Metode `resize()` memperbarui viewport agar tampilan tetap proporsional ketika ukuran layar berubah. Fungsi `show()`, `pause()`, `resume()`, dan `hide()` tidak membutuhkan logika tambahan sehingga dibiarkan kosong. Sementara itu, `dispose()` digunakan untuk membebaskan semua resource yang dipakai, seperti `stage`, `batch`, `font`, musik intro, texture gambar awal, serta seluruh texture intro yang dimuat ke dalam array. Dengan proses pembersihan ini, memori tetap aman dan game tidak mengalami kebocoran resource setelah intro selesai.

2.3 Challenge

Kami memiliki beberapa tantangan seperti berikut :

1. Program yang kami buat memiliki tingkat kompleksitas yang tinggi sehingga sulit untuk melakukan update maupun maintain kode dikarenakan perlu komunikasi yang baik.

2. Gaya dan minat belajar siswa yang berbeda-beda sehingga tidak semua siswa tertarik dengan program yang kami buat sehingga diperlukan desain yang menarik dan game dengan storytelling yang seru.
3. Keterbatasan teknologi dimana tidak semua siswa memiliki laptop untuk menjalankan program kami.

2.4 Opportunity

Pengembangan aplikasi pembelajaran sejarah berbasis video dan game interaktif ini memiliki peluang yang sangat besar untuk diterapkan maupun dikembangkan lebih lanjut. Di tengah meningkatnya penggunaan teknologi digital dalam dunia pendidikan, program ini berpotensi menjadi media pendukung pembelajaran yang inovatif dan efektif.

1. Pertama, aplikasi ini dapat menjadi alternatif pembelajaran sejarah yang lebih menarik bagi siswa yang cenderung memiliki gaya belajar visual dan kinestetik. Dengan menggabungkan animasi, video, dan game, siswa mendapatkan pengalaman belajar yang lebih menyenangkan dan imersif, sehingga peluang peningkatan minat belajar sangat terbuka.
2. Kedua, program ini dapat dikembangkan menjadi media edukasi berskala lebih luas, seperti bahan ajar digital resmi untuk sekolah, konten pembelajaran yang terintegrasi dengan Kurikulum Merdeka, hingga platform edukatif yang dapat digunakan oleh masyarakat umum. Potensi kolaborasi dengan guru, sekolah, atau lembaga pendidikan juga sangat besar karena kebutuhan akan media pembelajaran kreatif terus meningkat.
3. Ketiga, aplikasi ini membuka peluang pengembangan fitur tambahan seperti leaderboard, sistem poin edukatif, mini quiz, atau materi sejarah interaktif lain yang dapat meningkatkan retensi belajar siswa. Program juga dapat diperluas ke berbagai periode sejarah Indonesia, sehingga nilai edukatifnya semakin kaya.
4. Terakhir, perkembangan industri game edukasi yang semakin pesat menjadi kesempatan bagi proyek ini untuk memasuki pasar edtech secara lebih serius. Dengan peningkatan kualitas visual, gameplay, dan materi ajar, aplikasi ini dapat memiliki nilai komersial sekaligus menjadi kontribusi nyata dalam meningkatkan literasi sejarah generasi muda.

