UAS PBO
Project Game FlappyBird

Oleh :

Ferry Yudhistira (1911102441069)
M. Yogi Saputra (1911102441014)
M. Alif Nur Fajri (2111102441129)

Teknik Informatika
Fakultas Sains & Teknologi
Universitas Muhammadiyah Kalimantan Timur

Samarinda, 2023

Detail Proyek: Game dengan Konsep PBO/OOP

Deskripsi Tugas: Anda diminta untuk membuat sebuah game menggunakan platform Greenfoot yang menunjukkan penerapan konsep-konsep Pemrograman Berbasis Objek. Game ini dapat berupa permainan apa pun, seperti game platformer, puzzle, atau shooter, dan harus mencakup minimal beberapa elemen berikut:

1. **- Kelas dan Objek:** Penerapan kelas dan objek yang jelas dan sesuai dengan konsep PBO/OOP.

Kelas Bird:

- Representasi burung sebagai objek dalam game.
- Variabel anggota: posisi burung, kecepatan vertikal, status hidup/mati.
- Metode: act (untuk pembaruan setiap frame), lompat, turun, cek tabrakan.

Kelas Pipe:

- Representasi pipa sebagai objek dalam game.
- Variabel anggota: posisi pipa, lebar, tinggi, kecepatan pergerakan.
- Metode: act (untuk pembaruan setiap frame), respawn jika keluar dari layar.

Kelas FlappyWorld:

- Kelas utama yang mengelola objek-objek dalam permainan.
- Metode: act (untuk pembaruan setiap frame), spawn pipa, cek tabrakan, dsb.

2. **- Pewarisan (Inheritance):** Penggunaan pewarisan untuk membangun hubungan antar kelas yang logis dan efisien.

Kelas ActorIndestructible:

- Kelas ini mewakili objek dalam permainan yang tidak dapat dihancurkan, misalnya latar belakang atau elemen dekoratif.
- Kelas ini mewarisi dari **Actor** karena kita ingin objek ini dapat ditambahkan ke dunia Greenfoot.

Kelas Bird (warisan dari ActorIndestructible):

- Kelas ini mewakili burung dalam permainan.
- Mewarisi dari **ActorIndestructible** karena burung juga merupakan aktor di dunia permainan.

Kelas Pipe (warisan dari Actor):

- Kelas ini mewakili pipa dalam permainan.
- Mewarisi dari **Actor** karena pipa adalah aktor yang dapat dihancurkan oleh tabrakan.

Kelas ScorableActor (warisan dari Actor):

- Kelas ini mewakili objek dalam permainan yang dapat memberikan skor kepada pemain.
- Mewarisi dari **Actor** karena objek ini dapat dihancurkan oleh tabrakan.

Kelas Coin (warisan dari ScorableActor):

- Kelas ini mewakili koin dalam permainan.
- Mewarisi dari **ScorableActor** karena koin dapat memberikan skor kepada pemain.

3. **- Polimorfisme:** Pemanfaatan polimorfisme untuk menciptakan variasi dalam perilaku objek.

1. **Interface Scorable:**
    - Interface ini digunakan untuk objek yang dapat memberikan skor kepada pemain.

```java
public interface Scorable {
    int getScoreValue();
}
```

2. **Kelas Bird:**
    - Kelas ini mewakili burung dalam permainan dan mengimplementasikan interface **Scorable**.
    - Burung memberikan skor negatif jika bertabrakan dengan pipa.

```java
public class Bird extends ActorIndestructible implements Scorable {
    private int scoreValue;

    public Bird() {
        scoreValue = -10;
    }

    public int getScoreValue() {
        return scoreValue;
    }

    // Logika khusus untuk burung
}
```

3. **Kelas Pipe:**
    - Kelas ini mewakili pipa dalam permainan dan juga mengimplementasikan interface **Scorable**.
    - Pipa memberikan skor positif jika berhasil dilewati oleh burung.

```java
public class ScorableActor extends Actor implements Scorable {
    private int scoreValue;

    public void setScoreValue(int value) {
        scoreValue = value;
    }

    public int getScoreValue() {
        return scoreValue;
    }
}
```

4. **Kelas ScorableActor:**

- Kelas ini mewakili objek dalam permainan yang dapat memberikan skor kepada pemain dan mengimplementasikan interface **Scorable**.
- Kelas ini memungkinkan penambahan objek lain yang memberikan skor tanpa mengubah kelas **Bird** atau **Pipe**.

java Copy code

```java
public class ScorableActor extends Actor implements Scorable {
    private int scoreValue;

    public void setScoreValue(int value) {
        scoreValue = value;
    }

    public int getScoreValue() {
        return scoreValue;
    }
}
```

Dengan menggunakan polimorfisme dan interface, Anda dapat mengelola objek-objek dalam permainan dengan cara yang lebih umum dan fleksibel. Misalnya, ketika pemain mendeteksi tabrakan, Anda dapat memeriksa apakah objek tersebut adalah instance dari **Scorable**, dan jika ya, Anda dapat menambahkan skor sesuai. Hal ini memudahkan penambahan objek baru yang dapat memberikan skor tanpa harus mengubah implementasi kelas **Bird** atau **Pipe**.

4. **- Enkapsulasi:** Penggunaan enkapsulasi untuk melindungi data dan memastikan akses yang aman.

Dalam konteks game Flappy Bird, kita dapat menggunakan enkapsulasi untuk melindungi data seperti posisi, skor, atau status hidup burung dari akses langsung.

Berikut adalah contoh penggunaan enkapsulasi dalam kelas Bird:

```
}

// Metode khusus untuk melakukan lompat
public void jump() {
    setVelocity(-10);
}

// Metode khusus untuk melakukan update posisi
public void update() {
    setVelocity(getVelocity() + 1);
    setLocation(getX(), getY() + getVelocity());
}

// Metode khusus untuk melakukan pengecekan tabrakan
public void checkCollision(Pipe pipe) {
    // Logika deteksi tabrakan dengan pipa
}

public int getScoreValue() {
    return getScore();
}
```

```
// Metode get dan set untuk isAlive
public boolean isAlive() {
    return isAlive;
}

public void setAlive(boolean alive) {
    isAlive = alive;
}

// Metode get dan set untuk score
public int getScore() {
    return score;
}

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

// Metode khusus untuk melakukan lompat
public void jump() {
    setVelocity(-10);
```

```java
public class Bird extends ActorIndestructible implements Scorable {
    private int velocity;
    private boolean isAlive;
    private int score;

    public Bird() {
        velocity = 0;
        isAlive = true;
        score = 0;
    }

    // Metode get dan set untuk velocity
    public int getVelocity() {
        return velocity;
    }

    public void setVelocity(int velocity) {
        this.velocity = velocity;
    }
```

5. **- Interaksi Antar Objek:** Implementasi interaksi antar objek yang dinamis dan menarik.

Untuk menciptakan interaksi antar objek yang dinamis dan menarik dalam game Flappy Bird, Anda dapat memanfaatkan konsep deteksi tabrakan, perubahan status objek, dan tanggapan visual atau suara. Dalam kasus ini, kita akan fokus pada interaksi antara burung (**Bird**) dan pipa (**Pipe**).

Berikut adalah contoh implementasi interaksi antar objek dengan deteksi tabrakan dan beberapa efek visual dalam Greenfoot:

Kelas Bird:

```
        this.score = score;
    }

    public int getVelocity() {
        return velocity;
    }

    public void setVelocity(int velocity) {
        this.velocity = velocity;
    }

    public int getScoreValue() {
        return getScore();
    }
}
private void handleCollisionWithPipe(Pipe pipe) {
    setAlive(false);
    // Implementasi lainnya sesuai kebutuhan, misalnya menghentikan
}

public void setAlive(boolean alive) {
    isAlive = alive;
}

public int getScore() {
    return score;
}

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

public int getVelocity() {
    return velocity;
}
```

```
private void handleInput() {
    if (Greenfoot.isKeyDown("space")) {
        jump();
    }
}

private void jump() {
    setVelocity(-10);
}

private void update() {
    setVelocity(getVelocity() + 1);
    setLocation(getX(), getY() + getVelocity());
}

private void checkCollision() {
    Pipe pipe = (Pipe) getOneIntersectingObject(Pipe.class);
    if (pipe != null) {
        handleCollisionWithPipe(pipe);
    }
}
```

```
public class Bird extends ActorIndestructible implements Scorable {
    private int velocity;
    private boolean isAlive;
    private int score;

    public Bird() {
        velocity = 0;
        isAlive = true;
        score = 0;
    }

    public void act() {
        if (isAlive) {
            handleInput();
            update();
            checkCollision();
        }
    }
```

KelasPipe:

```
}

// Metode khusus untuk melakukan lompat
public void jump() {
    setVelocity(-10);
}

// Metode khusus untuk melakukan update posisi
public void update() {
    setVelocity(getVelocity() + 1);
    setLocation(getX(), getY() + getVelocity());
}

// Metode khusus untuk melakukan pengecekan tabrakan
public void checkCollision(Pipe pipe) {
    // Logika deteksi tabrakan dengan pipa
}

public int getScoreValue() {
    return getScore();
}
```

```
    private void checkCollisionWithBird() {
        Bird bird = (Bird) getOneIntersectingObject(Bird.class);
        if (bird != null) {
            handleCollisionWithBird(bird);
        }
    }

    private void handleCollisionWithBird(Bird bird) {
        // Logika penanganan tabrakan dengan burung, misalnya mengurang
        bird.setAlive(false);
    }

    public int getScoreValue() {
        return 5; // Skor yang diberikan jika burung berhasil melewati
    }
}
```

6. **- Overriding dan Overloading:** Penggunaan overriding untuk mengganti perilaku metode dari superclass dan overloading untuk membuat metode dengan parameter yang berbeda.

Overriding Metode act :

Dalam kelas **Bird**, kita bisa mengganti perilaku metode **act** yang berasal dari superclass **ActorIndestructible**.

```java
public class Bird extends ActorIndestructible implements Scorable {
    // ...

    @Override
    public void act() {
        if (isAlive) {
            handleInput();
            update();
            checkCollision();
            applyGravity(); // Overriding: menambah perilaku baru
        }
    }

    private void applyGravity() {
        // Logika penerapan gravitasi pada burung
    }

    // ...
}
```

Overloading Metode checkCollision:
Kita bisa membuat beberapa versi metode **checkCollision** dengan parameter yang berbeda (overloading) untuk menangani kollisi dengan objek lain.

```
public class Bird extends ActorIndestructible implements Scorable {
    // ...

    private void checkCollision() {
        Pipe pipe = (Pipe) getOneIntersectingObject(Pipe.class);
        if (pipe != null) {
            handleCollisionWithPipe(pipe);
        }
    }

    private void checkCollision(Coin coin) {
        if (isTouching(coin.getClass())) {
            handleCollisionWithCoin(coin);
        }
    }

    private void handleCollisionWithCoin(Coin coin) {
        setScore(getScore() + coin.getScoreValue());
        coin.removeFromWorld();
    }

    // ...
}
```

Overriding Metode handleCollisionWithPipe di kelas Pipe:
Di kelas **Pipe**, kita bisa mengganti perilaku metode **handleCollisionWithBird** yang berasal dari superclass **Pipe**.

```
public class Pipe extends Actor implements Scorable {
    // ...

    private void handleCollisionWithBird(Bird bird) {
        bird.setAlive(false);
        // Logika penanganan tabrakan dengan burung pada pipa
    }

    // ...
}
```

membuat variasi dalam perilaku metode yang memungkinkan fleksibilitas dan modularitas yang lebih baik dalam desain permainan.