**Tugas Akhir Proyek Greenfoot**

**KONSEP PEMROGRAMAN BERORIENTASI OBJEK**

# DOKUMENTASI

# GAME GATOT RIDER


**OLEH:**

**412023011 - Steven Felizio**

**412023037 – Dhian Juwita Putri**

**PROGRAM STUDI INFORMATIKA**
**FAKULTAS TEKNIK DAN ILMU KOMPUTER**
**UNIVERSITAS KRISTEN KRIDA WACANA**
**JAKARTA**


**2024**

# DAFTAR ISI

# KATA PENGANTAR

Puji syukur dipanjatkan ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya laporan dokumentasi yang berjudul DOKUMENTASI GAME GATOTRIDER dapat diselesaikan dengan baik dan tepat waktu. Laporan dokumentasi ini disusun sebagai salah satu tugas mata kuliah Konsep Pemrograman Berorientasi Objek dan bertujuan untuk mengkaji implementasi bahasa pemrograman Java pada aplikasi Greenfoot untuk mengembangkan sebuah *game* dua dimensi dengan tema "Kekayaan potensi daerah Indonesia". Target pengguna aplikasi *game* adalah siswa kelas 2 Sekolah Dasar dengan harapan melatih keterampilan *High Order Thinking Skill* (HOTS), kemampuan berhitung, pengenalan alfabet, dan penggunaan *mouse* dan *keyboard* siswa. Laporan dokumentasi ini diharapkan dapat memberikan gambaran dalam pengembangan *game* GatotRider dan menjadi referensi bagi pembelajaran berikutnya. Akhir kata, semoga laporan dokumentasi ini dapat bermanfaat dan memberikan kontribusi positif dalam pembelajaran mata kuliah konsep pemrograman berorientasi objek.

Jakarta, 23 Agustus 2024

**Dhian Juwita Putri / Steven Felizio**

# PENDAHULUAN

Laporan dokumentasi ini disusun sebagai bagian dari pengembangan aplikasi permainan *GatotRider,* yang merupakan implementasi dari mata kuliah Konsep Pemrograman Berorientasi Objek. *Game* ini dirancang dan dikembangkan menggunakan platform *Greenfoot*, yang memungkinkan aplikasi *game* dapat dimainkan dengan baik melalui situs web *Greenfoot*.

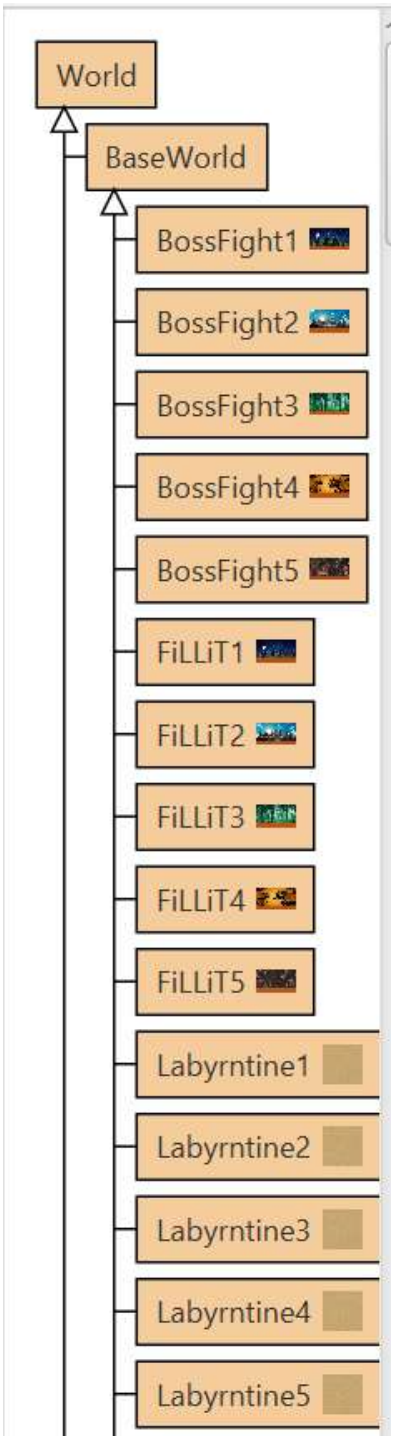
Aplikasi *game GatotRider* mengusung tema Kekayaan Potensi Daerah Indonesia, yaitu seorang tokoh pewayangan yang populer di Indonesia bernama Gatot Kaca. Permainan ini ditargetkan untuk siswa kelas 2 Sekolah Dasar, dengan tujuan memberikan pengalaman belajar yang menyenangkan sekaligus mendidik.

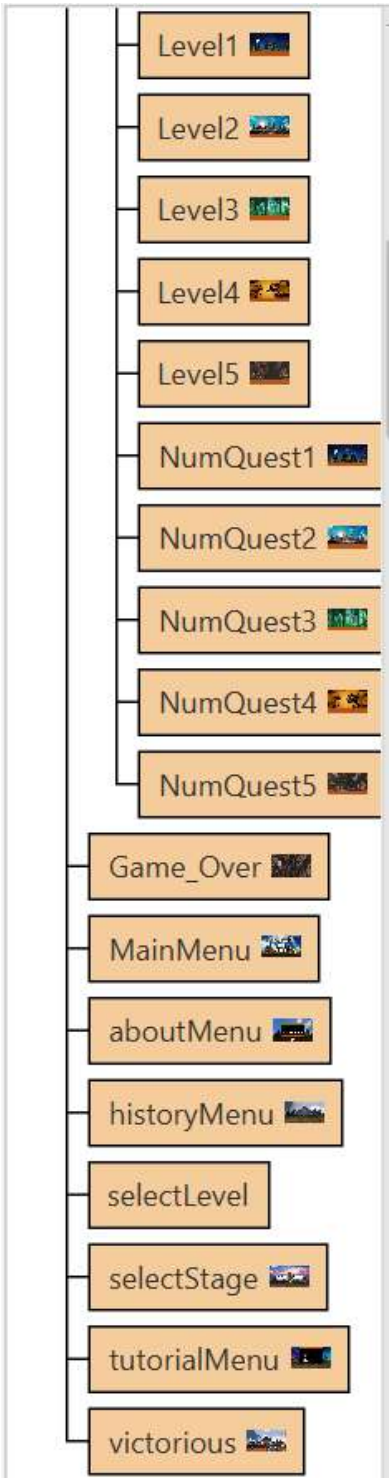
Melalui *game GatotRider*, diharapkan dapat melatih keterampilan berpikir tingkat tinggi (*Higher Order Thinking Skills/HOTS*) siswa, memperkuat kemampuan berhitung dalam operasi penjumlahan dan pengurangan, mengenalkan huruf secara interaktif, serta meningkatkan keterampilan dasar dalam penggunaan *mouse* dan *keyboard*.

Laporan ini akan menjelaskan proses pengembangan permainan *GatotRider*, yang tercakup dalam modul pembuatan dan dokumentasi teknis. Dokumentasi ini juga mencakup detail tentang fitur, antarmuka pengguna, mekanisme permainan, dan penjelasan mengenai program dalam *game*. Diharapkan laporan ini dapat bermanfaat dan memberikan kontribusi positif dalam pembelajaran mata kuliah konsep pemrograman berorientasi objek.

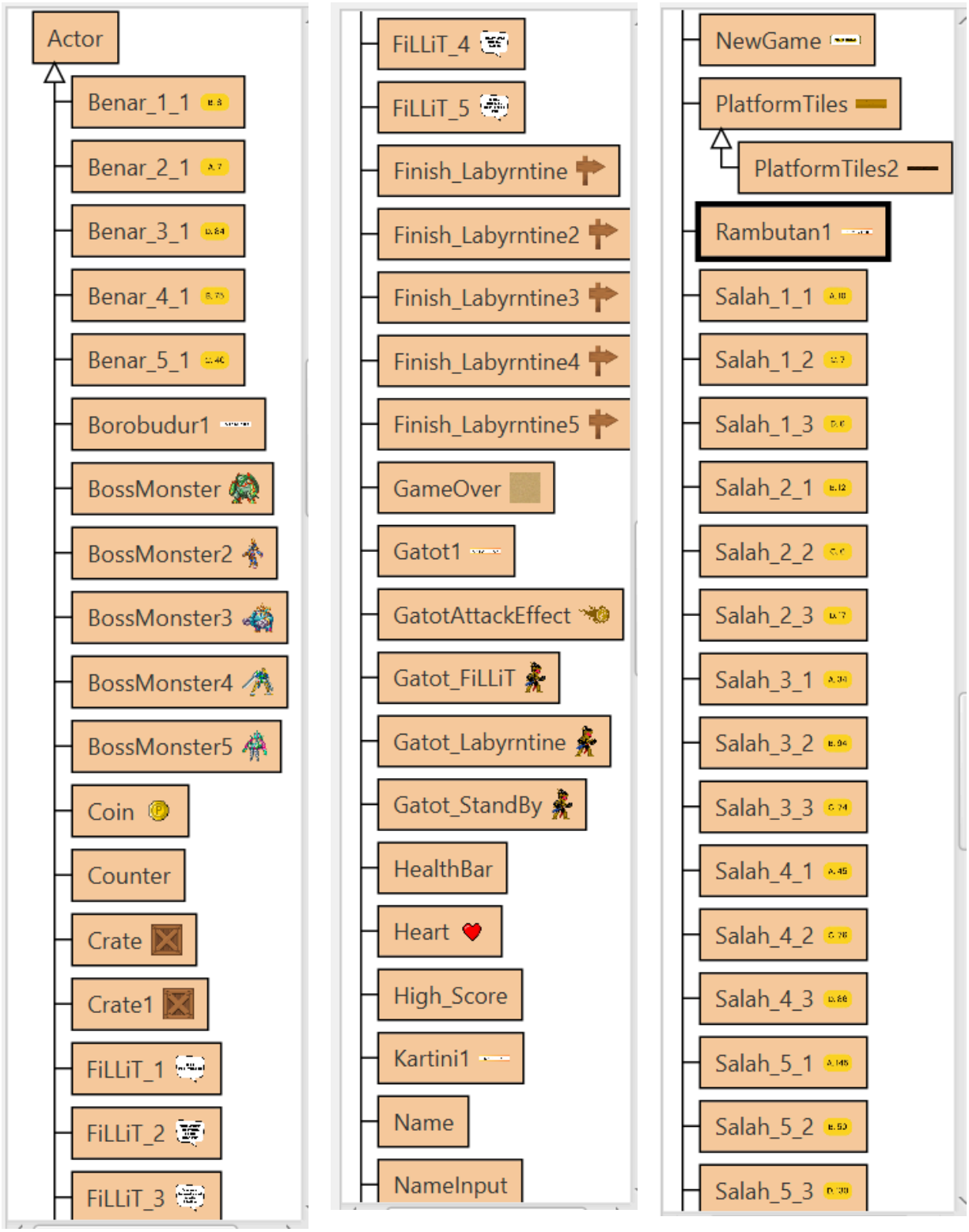
**PENGERJAAN KODE JAVA PADA GREENFOOT**

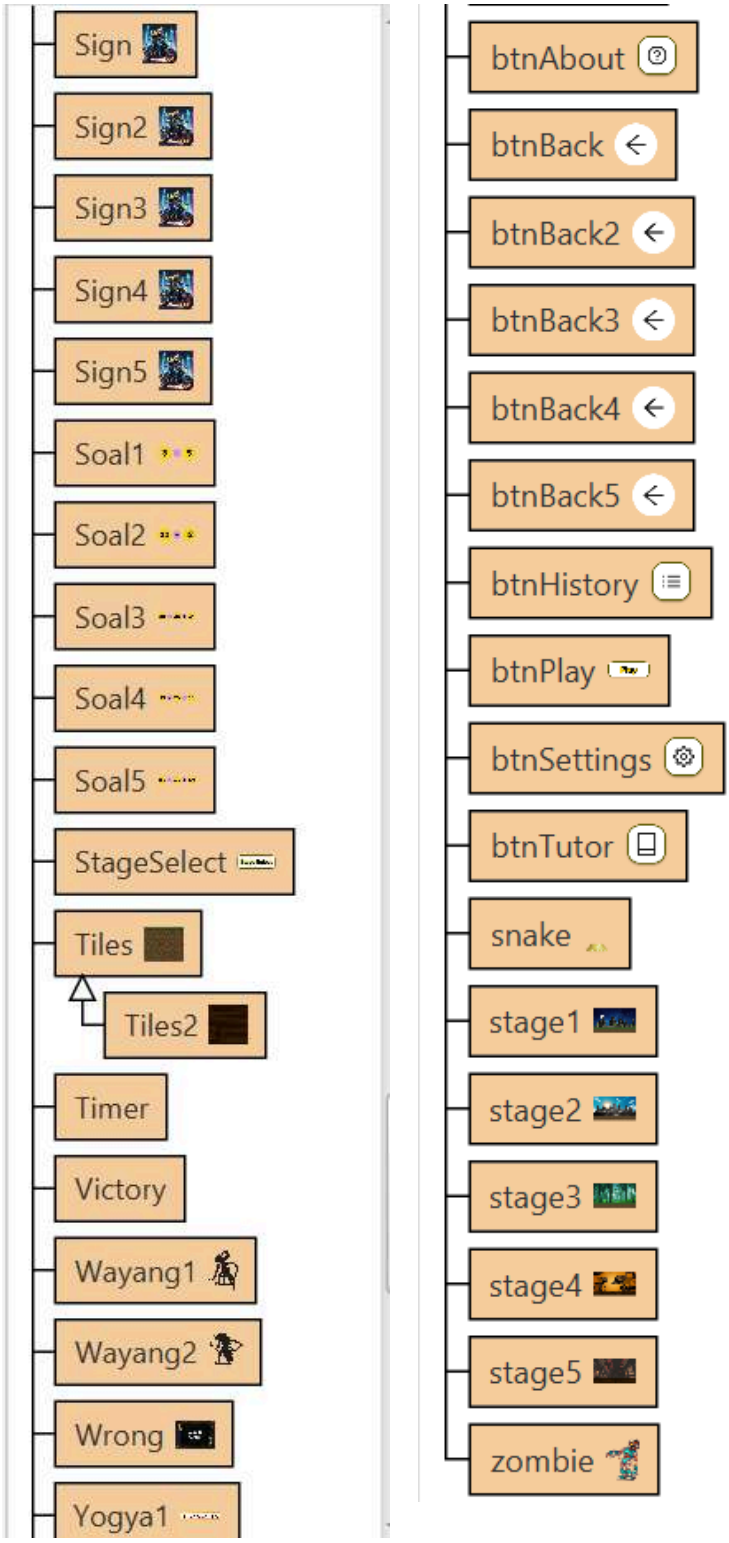
World :



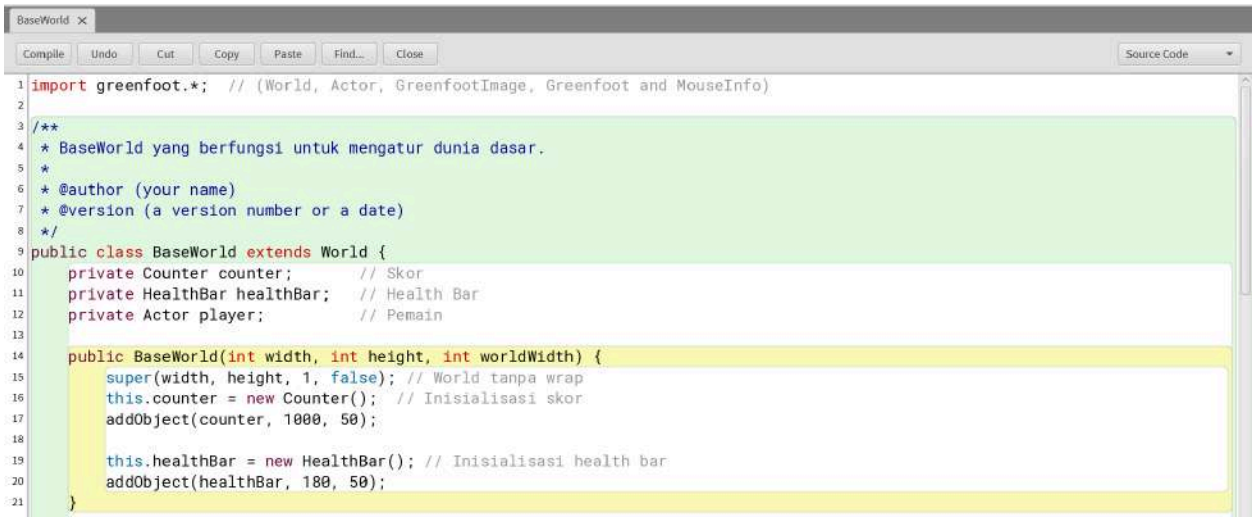



Actor :


Actor
Benar_1_1
Benar_2_1
Benar_3_1
Benar_4_1
Benar_5_1
Borobudur1
BossMonster
BossMonster2
BossMonster3
BossMonster4
BossMonster5
Coin
Counter
Crate
Crate1
FiLLiT_1
FiLLiT_2
FiLLiT_3
FiLLiT_4
FiLLiT_5
Finish_Labyrntine
Finish_Labyrntine2
Finish_Labyrntine3
Finish_Labyrntine4
Finish_Labyrntine5
GameOver
Gatot1
GatotAttackEffect
Gatot_FiLLiT
Gatot_Labyrntine
Gatot_StandBy
HealthBar
Heart
High_Score
Kartini1
Name
NameInput
NewGame
PlatformTiles
PlatformTiles2
Rambutan1
Salah_1_1
Salah_1_2
Salah_1_3
Salah_2_1
Salah_2_2
Salah_2_3
Salah_3_1
Salah_3_2
Salah_3_3
Salah_4_1
Salah_4_2
Salah_4_3
Salah_5_1
Salah_5_2
Salah_5_3


Sign
Sign2
Sign3
Sign4
Sign5
Soal1
Soal2
Soal3
Soal4
Soal5
StageSelect
Tiles
Tiles2
Timer
Victory
Wayang1
Wayang2
Wrong
Yogya1
btnAbout
btnBack
btnBack2
btnBack3
btnBack4
btnBack5
btnHistory
btnPlay
btnSettings
btnTutor
snake
stage1
stage2
stage3
stage4
stage5
zombie

**BaseWorld**




```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * BaseWorld yang berfungsi untuk mengatur dunia dasar.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BaseWorld extends World {
    private Counter counter;       // Skor
    private HealthBar healthBar;   // Health Bar
    private Actor player;          // Pemain

    public BaseWorld(int width, int height, int worldWidth) {
        super(width, height, 1, false); // World tanpa wrap
        this.counter = new Counter();  // Inisialisasi skor
        addObject(counter, 1000, 50);

        this.healthBar = new HealthBar(); // Inisialisasi health bar
        addObject(healthBar, 180, 50);
    }
```

Penjelasan :

```
private Counter counter;       // Skor
private HealthBar healthBar;   // Health Bar
private Actor player;          // Pemain
```

> Membuat atribut untuk mencatat skor (Counter), health (HealthBar), dan referensi pemain (Actor).

```
public BaseWorld(int width, int height, int worldWidth) {
      super(width, height, 1, false); // World tanpa wrap
      this.counter = new Counter();  // Inisialisasi skor
      addObject(counter, 1000, 50);

      this.healthBar = new HealthBar(); // Inisialisasi health bar
      addObject(healthBar, 180, 50);
      }
```

> Membuat dunia dasar tanpa efek wrap (dunia game akan mengikuti).

> Menambahkan skor (Counter) pada posisi (1000, 50) - skor akan terus berada di posisi ini.

> Menambahkan health bar (HealthBar) pada posisi (180, 50) - health akan terus berada di posisi ini.

```
    protected void setPlayer(Actor player, int startX, int startY) {
        this.player = player;
        addObject(player, startX, startY);
    }

    protected Actor getPlayer() {
        return player;
    }

    public Counter getCounter() {
        return counter;
    }

    public HealthBar getHealthBar() {
        return healthBar;
    }

    protected void keepCounterSticky() {
        if (counter != null) {
            counter.setLocation(1000, 50);
        }
    }

    protected void keepHealthBarSticky() {
        if (healthBar != null) {
            healthBar.setLocation(180, 50);
        }
    }
}
```

Penjelasan :

```
protected void setPlayer(Actor player, int startX, int startY) {
    this.player = player;
    addObject(player, startX, startY);
}
```

> Menempatkan pemain (Gatot_StandBy) di koordinat (startX, startY) sesuai kebutuhan level (Rata-rata di setiap level penempatannya sama).

```
protected Actor getPlayer() {
    return player;
}
```

> Mengembalikan referensi pemain saat ini.

NOTE : protected digunakan untuk membatasi penggunaan method hanya pada package yang sama (dalam kode ini yaitu untuk world turunan lain), bedanya dengan public ialah kalau public dapat diakses secara global dimana saja.

```
public Counter getCounter() {
    return counter;
}

public HealthBar getHealthBar() {
    return healthBar;
}
```

> Mengembalikan objek skor (Counter) dan HealthBar untuk digunakan atau dimodifikasi lebih lanjut.

```
protected void keepCounterSticky() {
        if (counter != null) {
            counter.setLocation(1000, 50);
        }
    }

    protected void keepHealthBarSticky() {
        if (healthBar != null) {
            healthBar.setLocation(180, 50);
        }
    }
```

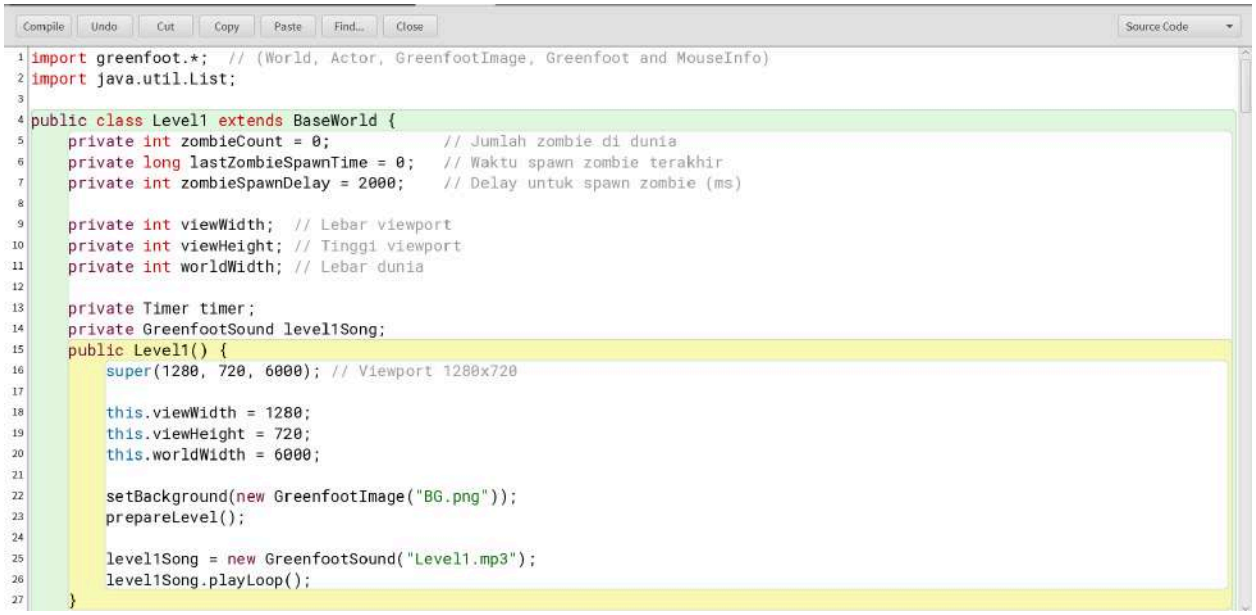
> Menjaga posisi actor Counter dan HealthBar tetap pada lokasi yang ditentukan.

**World Stages**

**Terdapat 5 level, setiap level ada 5 world yang harus dilewati. Semua world ini berada dalam BaseWorld (Extends BaseWorld) untuk mengambil health dan skor.**

**World dimana player melawan hydra enemies terdiri dari Level1, Level2, Level3, Level4, dan Level5.**

**Level1**



```
    public void act() {
        // Logika kamera untuk mengikuti pemain
        followPlayerWithCamera(); // Fokus kamera pada pemain
        spawnZombiesWithDelay();  // Spawn zombie
        keepCounterSticky();      // Jaga posisi skor tetap
        keepHealthBarSticky();    // Jaga posisi health bar tetap
        keepTimerSticky();
    }

    @Override
    public void stopped() {
    level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level1Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level1Song.stop();
    }
```

```
    private void prepareLevel() {
        // player
        Gatot_StandBy player = new Gatot_StandBy(90, 130);
        setPlayer(player, 100, 459); // Posisi awal pemain

        // tiles
        createGroundTiles(128, 651);

        // sign
        addObject(new Sign(), 6000, 459); // Tanda akhir level

        // zombie
        spawnInitialZombies();

        // snake
        spawnInitialSnakes();

        // snake random
        spawnMultipleSnakes(5);

        //timer
        timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        // platform
        PlatformTiles platformTiles = new PlatformTiles();
        addObject(platformTiles,1200, 416);
```

```

        PlatformTiles platformTiles1 = new PlatformTiles();
        addObject(platformTiles1,1270, 416);

        // coin
        Coin coin1 = new Coin();
        coin1.getImage().scale(100, 100);
        addObject(coin1, 1450, 210);

        PlatformTiles platformTiles2 = new PlatformTiles();
        addObject(platformTiles2,1450, 270);

        PlatformTiles platformTiles3 = new PlatformTiles();
        addObject(platformTiles3,2500, 416);

        PlatformTiles platformTiles4 = new PlatformTiles();
        addObject(platformTiles4,2570, 416);

        Heart heart = new Heart();
        heart.getImage().scale(100, 100);
        addObject(heart, 2850, 180);

        PlatformTiles platformTiles5 = new PlatformTiles();
        addObject(platformTiles5,2850, 265);

```

```
        PlatformTiles platformTiles6 = new PlatformTiles();
        addObject(platformTiles6,3200, 416);

        PlatformTiles platformTiles7 = new PlatformTiles();
        addObject(platformTiles7,4000, 369);

        PlatformTiles platformTiles8 = new PlatformTiles();
        addObject(platformTiles8,4500, 341);

        //crate1
        Crate1 crate1 = new Crate1();
        addObject(crate1,2603,357);

        Crate1 crate12 = new Crate1();
        addObject(crate12,4200,256);

        Heart heart1 = new Heart();
        heart1.getImage().scale(100, 100);
        addObject(heart1, 4350, 50);

        Crate1 crate13 = new Crate1();
        addObject(crate13,4350,142);

    }
```

**NOTE :** .scale digunakan untuk menyesuaikan width dan height dari sebuah objek sesuai kebutuhan.

```
    private void spawnInitialZombies() {
        zombie zombie1 = new zombie();
        addObject(zombie1, 894, 526);

        zombie zombie2 = new zombie();
        addObject(zombie2, 1100, 526);

        zombie zombie3 = new zombie();
        addObject(zombie3, 1500, 526);

        zombie zombie4 = new zombie();
        addObject(zombie4, 2600, 526);

        zombie zombie5 = new zombie();
        addObject(zombie5, 2000, 526);

        zombie zombie6 = new zombie();
        addObject(zombie6, 1900, 526);
    }
```

```
    private void spawnInitialSnakes() {
        snake snake1 = new snake();
        addObject(snake1,899,551);

        snake snake2 = new snake();
        addObject(snake2,1000,551);

        snake snake3 = new snake();
        addObject(snake3,1200,551);
    }

    private void createGroundTiles(int tileWidth, int groundY) {
        int numTiles = (int) Math.ceil((double) 6000 / tileWidth); // 6000 lebar dunia
        for (int i = 0; i < numTiles; i++) {
            Tiles tile = new Tiles();
            addObject(tile, i * tileWidth, groundY);
        }
    }
```

```
166     private void spawnMultipleSnakes(int count) {
167         List<Tiles> tiles = getObjects(Tiles.class);
168 
169         for (int i = 0; i < count; i++) {
170             if (!tiles.isEmpty()) {
171                 int snakeX;
172                 int snakeY;
173                 Tiles randomTile;
174 
175                 do {
176                     randomTile = tiles.get(Greenfoot.getRandomNumber(tiles.size()));
177                     snakeX = randomTile.getX();
178                 } while (snakeX <= 130);
179 
180                 snakeY = randomTile.getY() - 100;
181 
182                 snake snake = new snake();
183                 addObject(snake, snakeX, snakeY);
184             }
185         }
186     }
187 
188     private void addZombie(int x, int y) {
189         zombie newZombie = new zombie();
190         addObject(newZombie, x, y);
191         zombieCount++;
192     }
```

```
194     private void spawnZombiesWithDelay() {
195         int zombieX = 3980; // Posisi spawn zombie
196         int groundY = 526;  // Posisi zombie di atas ground
197 
198         if (zombieCount < 8 && System.currentTimeMillis() - lastZombieSpawnTime >= zombieSpawnDelay) {
199             addZombie(zombieX, groundY);
200             lastZombieSpawnTime = System.currentTimeMillis();
201         }
202     }
```

```
203 
204     private void followPlayerWithCamera() {
205         if (getPlayer() == null) return;
206 
207         // Posisi pemain
208         int playerX = getPlayer().getX();
209         int playerY = getPlayer().getY();
210 
211         // posisi kamera, pemain mid
212         int viewX = Math.max(viewWidth / 2, Math.min(playerX, worldWidth - viewWidth / 2));
213         int offsetX = viewX - viewWidth / 2;
214 
215         // geser seluruh dunia berdasarkan offsetX
216         for (Object obj : getObjects(null)) {
217             Actor actor = (Actor) obj;
218             int newX = actor.getX() - offsetX;
219             actor.setLocation(newX, actor.getY());
220         }
221 
222         // supaya ga keluar viewport
223         int limitedPlayerX = Math.max(getPlayer().getImage().getWidth() / 2,
224                 Math.min(playerX, worldWidth - getPlayer().getImage().getWidth() / 2));
225         if (playerX != limitedPlayerX) {
226             getPlayer().setLocation(limitedPlayerX, playerY);
227         }
228     }
```

```
    protected void keepCounterSticky() {
        if (getCounter() != null) {
            getCounter().setLocation(getCameraOffsetX() + 1000, 50);
        }
    }

    protected void keepHealthBarSticky() {
        if (getHealthBar() != null) {
            getHealthBar().setLocation(getCameraOffsetX() + 180, 50);
        }
    }

    // Sticky for the timer
    private void keepTimerSticky() {
        if (timer != null) {
            timer.setLocation(getCameraOffsetX() + 1180, 50);  // Keep the timer at the same position
        }
    }

    private int getCameraOffsetX() {
        if (getPlayer() == null) return 0;

        int playerX = getPlayer().getX();
        return Math.max(0, Math.min(playerX - viewWidth / 2, worldWidth - viewWidth));
    }
}
```

Penjelasan :

import java.util.List;

> Mengimpor kelas List dari paket java.util. Bekerja untuk koleksi data yang terurut dan dapat diakses berdasarkan indeks.

```
    private int zombieCount = 0;            // Jumlah zombie di dunia
    private long lastZombieSpawnTime = 0;   // Waktu spawn zombie terakhir
    private int zombieSpawnDelay = 2000;    // Delay untuk spawn zombie (ms)

    private int viewWidth;  // Lebar viewport
    private int viewHeight; // Tinggi viewport
    private int worldWidth; // Lebar dunia

    private Timer timer;
    private GreenfootSound level1Song;
```

> zombieCount: Menghitung jumlah zombie untuk yang akan dipakai pada spawnZombiesWithDelay.

> lastZombieSpawnTime: Menyimpan waktu terakhir kali zombie muncul.

> zombieSpawnDelay: Jeda waktu (dalam milidetik) sebelum zombie baru muncul.

> viewWidth dan viewHeight: Menyimpan dimensi viewport (logika kamera).

> worldWidth: Menyimpan lebar dunia asli untuk diletakkan platforming dan enemies.

> timer: Objek penghitung waktu yang ditampilkan di layar.

> level1Song: Musik untuk Level1.

```
public Level1() {
        super(1280, 720, 6000); // Viewport 1280x720

        this.viewWidth = 1280;
        this.viewHeight = 720;
        this.worldWidth = 6000;

        setBackground(new GreenfootImage("BG.png"));
        prepareLevel();

        level1Song = new GreenfootSound("Level1.mp3");
        level1Song.playLoop();
    }
```

> super(1280, 720, 6000): Memanggil konstruktor dari BaseWorld untuk membuat dunia dengan viewport 1280x720 dan lebar dunia asli 6000.
> setBackground: Menetapkan gambar latar dunia.
> prepareLevel(): Metode untuk menambahkan objek-objek ke dunia.
> level1Song: Memutar lagu Level1.mp3 secara berulang-ulang selama player berada di world.

```
public void act() {
        // Logika kamera untuk mengikuti pemain
        followPlayerWithCamera(); // Fokus kamera pada pemain
        spawnZombiesWithDelay();  // Spawn zombie
        keepCounterSticky();      // Jaga posisi skor tetap
        keepHealthBarSticky();    // Jaga posisi health bar tetap
        keepTimerSticky();
    }
```

> Dalam method act ini maka world akan bertindak sesuai method-method di bawah yang dipanggil.
> followPlayerWithCamera(): Kamera mengikuti pemain saat bergerak secara horizontal.
> spawnZombiesWithDelay(): Memunculkan zombie baru dengan jeda waktu tertentu.
> keepCounterSticky(), keepHealthBarSticky(), dan keepTimerSticky(): Memastikan skor, health bar, dan timer tetap di posisi yang benar pada layar meskipun kamera bergerak dengan width asli yang berbeda-beda.

```
@Override
public void stopped() {
level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level1Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level1Song.stop();
}
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

```
private void prepareLevel() {
        …..;
}
private void spawnInitialZombies() {
        …..;
}
private void spawnInitialSnakes() {
        …..;
}
```

> Penempatan objek yang dibutuhkan dalam world Level1 (Gatot_StandBy, Platform, Tiles, Coin, Heart, Timer, serta tambahan snake dan zombie.

```
private void createGroundTiles(int tileWidth, int groundY) {
    int numTiles = (int) Math.ceil((double) 6000 / tileWidth); // 4000 lebar dunia
    for (int i = 0; i < numTiles; i++) {
        Tiles tile = new Tiles();
        addObject(tile, i * tileWidth, groundY);
    }
}
```

> Mengulang sebanyak jumlah tiles yang diperlukan untuk menutupi lebar dunia. Setiap iterasi, membuat tiles baru dan meletakkannya secara berurutan sepanjang sumbu X. numTiles dihitung dari pembagian lebar dunia (6000) dengan lebar tiles (tileWidth). Setiap tiles ditambahkan dengan posisi horizontal i * tileWidth dan vertikal tetap di groundY.

```
private void spawnMultipleSnakes(int count) {
    List<Tiles> tiles = getObjects(Tiles.class);

    for (int i = 0; i < count; i++) {
      if (!tiles.isEmpty()) {
        int snakeX;
        int snakeY;
        Tiles randomTile;

        do {
          randomTile = tiles.get(Greenfoot.getRandomNumber(tiles.size()));
          snakeX = randomTile.getX();
        } while (snakeX <= 130);

        snakeY = randomTile.getY() - 100;

        snake snake = new snake();
        addObject(snake, snakeX, snakeY);
      }
    }
  }
```

> Menambahkan beberapa ular ke dunia dengan jumlah yang ditentukan (count). Metode ini mengambil semua ubin (Tiles) yang ada, lalu dalam loop, memilih tiles secara acak, memastikan posisi X ubin lebih besar dari 130, dan menentukan posisi ular di atas tiles tersebut (Y dikurangi 100). Setelah itu, objek ular dibuat dan ditambahkan ke dunia pada posisi yang dihitung. Proses ini diulangi hingga semua ular berhasil ditambahkan.

```
private void addZombie(int x, int y) {
    zombie newZombie = new zombie();
    addObject(newZombie, x, y);
    zombieCount++;
  }
```

> Membuat objek zombie baru dan menambahkannya ke dunia di posisi yang diberikan (X, Y). Setelah itu, penghitung zombie (zombieCount) ditambah 1 (Berhubungan dengan method penambahan zombie lain).

```
private void spawnZombiesWithDelay() {
        int zombieX = 3980; // Posisi spawn zombie
        int groundY = 526;  // Posisi zombie di atas ground

            if (zombieCount < 8 && System.currentTimeMillis() - lastZombieSpawnTime >=
zombieSpawnDelay) {
            addZombie(zombieX, groundY);
            lastZombieSpawnTime = System.currentTimeMillis();
        }
    }
```

> Memeriksa apakah jumlah zombie selain dari spawnInitialZombies() yang ada kurang dari 8 dan apakah waktu tunda spawn zombie telah tercapai. Jika kondisi terpenuhi, metode addZombie dipanggil untuk menambahkan zombie baru ke posisi yang telah ditentukan. Waktu spawn terakhir diperbarui setelah setiap spawn.

```
private void followPlayerWithCamera() {
        if (getPlayer() == null) return;

        // Posisi pemain
        int playerX = getPlayer().getX();
        int playerY = getPlayer().getY();

        // posisi kamera, pemain mid
        int viewX = Math.max(viewWidth / 2, Math.min(playerX, worldWidth - viewWidth / 2));
        int offsetX = viewX - viewWidth / 2;

        // geser seluruh dunia berdasarkan offsetX
        for (Object obj : getObjects(null)) {
            Actor actor = (Actor) obj;
            int newX = actor.getX() - offsetX;
            actor.setLocation(newX, actor.getY());
        }

        // supaya ga keluar viewport
        int limitedPlayerX = Math.max(getPlayer().getImage().getWidth() / 2,
                Math.min(playerX, worldWidth - getPlayer().getImage().getWidth() / 2));
        if (playerX != limitedPlayerX) {
            getPlayer().setLocation(limitedPlayerX, playerY);
        }
    }
```

> Menggerakkan kamera untuk mengikuti pemain saat bergerak di dunia. Kamera disesuaikan agar posisi pemain selalu berada di tengah layar, dan dunia digeser berdasarkan offset untuk menyesuaikan dengan posisi pemain. Kamera juga membatasi pergerakan pemain agar tetap dalam batas dunia.
> if (getPlayer() == null) return; Memeriksa apakah pemain ada di dunia game. Jika tidak ada, metode ini akan berhenti (return) dan tidak melakukan apa-apa.
> Posisi pemain diambil menggunakan getX() dan getY(). Ini memberikan koordinat posisi pemain dalam dunia game.
> viewX adalah posisi X kamera. Kamera harus selalu menempatkan pemain di tengah layar, tetapi tidak boleh keluar dari batas dunia game. Untuk itu, viewX dihitung dengan memastikan posisi pemain (playerX) berada dalam batas dunia, dengan kamera setengah layar di kiri dan kanan.
> Math.min(playerX, worldWidth - viewWidth / 2) memastikan kamera tidak melebihi batas dunia (membatasi playerX agar tidak lebih dari dunia minus setengah lebar layar).
> Math.max(viewWidth / 2, ...) memastikan kamera tidak bergerak lebih jauh dari batas kiri dunia (agar pemain tidak terlalu dekat dengan tepi kiri layar).
> offsetX adalah seberapa banyak dunia harus digeser untuk memastikan pemain berada di tengah layar. offsetX dihitung dengan mengurangi setengah lebar layar dari viewX.
> Seluruh objek dalam dunia digeser berdasarkan offsetX. Untuk setiap objek (actor) yang ada, posisi X-nya dikurangi dengan offsetX. Ini memindahkan objek-objek tersebut seiring dengan pergerakan kamera. Dunia game bergerak mengikuti posisi pemain.
> limitedPlayerX menghitung posisi X yang dibatasi agar pemain tidak lebih dari setengah lebar layar dari tepi dunia. Jika posisi pemain (playerX) tidak sama dengan limitedPlayerX, pemain dipindahkan ke posisi yang dibatasi ini.

```
protected void keepCounterSticky() {
        if (getCounter() != null) {
            getCounter().setLocation(getCameraOffsetX() + 1000, 50);
        }
    }

    protected void keepHealthBarSticky() {
        if (getHealthBar() != null) {
            getHealthBar().setLocation(getCameraOffsetX() + 180, 50);
        }
    }

    // Sticky for the timer
    private void keepTimerSticky() {
        if (timer != null) {
```

```
            timer.setLocation(getCameraOffsetX() + 1180, 50);
        }
    }
```
> Menjaga posisi HealthBar, timer, dan Counter tetap konsisten di layar dengan menyesuaikan posisinya berdasarkan offset kamera, memastikan health bar tetap terlihat pada layar saat pemain bergerak.

```
    private int getCameraOffsetX() {
        if (getPlayer() == null) return 0;

        int playerX = getPlayer().getX();
        return Math.max(0, Math.min(playerX - viewWidth / 2, worldWidth - viewWidth));
    }
```
> Menghitung seberapa banyak kamera perlu digeser secara horizontal agar pemain tetap berada di tengah layar. Jika pemain ada (dicek dengan getPlayer() != null), maka metode ini mengambil posisi X pemain menggunakan getX(). Math.max(0, ...) memastikan kamera tidak bergerak ke kiri melewati batas dunia. Hasil perhitungan ini adalah offsetX, yaitu seberapa banyak dunia harus digeser agar posisi pemain tetap terlihat di tengah layar. Metode ini sangat terkait dengan followPlayerWithCamera(), karena getCameraOffsetX() memberikan nilai offsetX yang digunakan untuk menggeser semua objek di dunia agar kamera mengikuti pemain.

**Level2, Level3, Level4, dan Level5 memiliki kode Java yang sama seperti Level1, perbedaannya hanya terdapat pada :**

1. Lebar dunia asli
    - Level2 : this.worldWidth = 10000;
    - Level3 : this.worldWidth = 15000;
    - Level4 : this.worldWidth = 20000;
    - Level5 : this.worldWidth = 25000;
2. Musik
    - Level2 : level2Song = new GreenfootSound("Level2.mp3");
    - Level3 : level3Song = new GreenfootSound("Level3.mp3");
    - Level4 : level4Song = new GreenfootSound("Level4.mp3");
    - Level5 : level5Song = new GreenfootSound("th06_06.mp3");
3. Set platform, coin, crate1, dan heart  (Level design)
    - Level2 : Set 1 - Set 10
    - Level3 : Set 1 - Set 15
    - Level4 : Set 1 - Set 20

- Level5 : Set 1 - Set 25

4. Jumlah zombie dan snake yang spawn
   Semakin tinggi level, maka semakin banyak zombie dan snake yang ditempatkan dan yang dispawn.
5. Background
   - Level2 : setBackground(new GreenfootImage("BG2.png"));
   - Level3 : setBackground(new GreenfootImage("BG3.png"));
   - Level4 : setBackground(new GreenfootImage("BG4.png"));
   - Level5 : setBackground(new GreenfootImage("BG5.png"));

**Level2**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;

public class Level2 extends BaseWorld {
    private int zombieCount = 0;            // Jumlah zombie di dunia
    private long lastZombieSpawnTime = 0;   // Waktu spawn zombie terakhir
    private int zombieSpawnDelay = 2000;    // Delay untuk spawn zombie (ms)

    private int viewWidth;  // Lebar viewport
    private int viewHeight; // Tinggi viewport
    private int worldWidth; // Lebar dunia

    private Timer timer;
    private GreenfootSound level2Song;
    public Level2() {
        super(1280, 720, 4000); // Viewport 1280x720, dunia 4000 px

        this.viewWidth = 1280;
        this.viewHeight = 720;
        this.worldWidth = 10000;

        setBackground(new GreenfootImage("BG2.png"));
        prepareLevel();

        level2Song = new GreenfootSound("Level2.mp3");
        level2Song.playLoop();
    }
```

```
public void act() {
    // Logika kamera untuk mengikuti pemain
    followPlayerWithCamera(); // Fokus kamera pada pemain
    spawnZombiesWithDelay();  // Spawn zombie
    keepCounterSticky();      // Jaga posisi skor tetap
    keepHealthBarSticky();    // Jaga posisi health bar tetap
    keepTimerSticky();
}

@Override
public void stopped() {
level2Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level2Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level2Song.stop();
}
```

```
private void prepareLevel() {
    // player
    Gatot_StandBy player = new Gatot_StandBy(90, 130);
    setPlayer(player, 100, 459); // Posisi awal pemain

    // tiles
    createGroundTiles(128, 651);

    // sign
    addObject(new Sign2(), 10000, 459); // Tanda akhir level

    // zombie
    spawnInitialZombies();

    // snake
    spawnInitialSnakes();

    // snake random
    spawnMultipleSnakes(10);

    //timer
    timer = new Timer();  // Initialize the timer
    addObject(timer, 1180, 50);  // Add the timer to the world

    // Set 1: Awal level - platform rendah dan Crate bertingkat
    PlatformTiles platform1 = new PlatformTiles();
    addObject(platform1, 400, 400); // Platform rendah dekat awal

    Crate1 crate1 = new Crate1();
    addObject(crate1, 600, 407); // Crate di tanah
```

```
Crate1 crate2 = new Crate1();
addObject(crate2, 600, 330); // Crate bertingkat untuk meloncat lebih tinggi

Coin coin1 = new Coin();
coin1.getImage().scale(100, 100);
addObject(coin1, 600, 170); // Coin di atas Crate bertingkat

Heart heart1 = new Heart();
heart1.getImage().scale(100, 100);
addObject(heart1, 800, 300); // Heart pertama di awal, mudah dicapai

// Set 2: Gabungan Platform tinggi dan Crate bertingkat
PlatformTiles platform2 = new PlatformTiles();
addObject(platform2, 1200, 380); // Platform sedang

Crate1 crate3 = new Crate1();
addObject(crate3, 1400, 407); // Crate rendah

Crate1 crate4 = new Crate1();
addObject(crate4, 1400, 330); // Crate tinggi di atas crate rendah

Coin coin2 = new Coin();
coin2.getImage().scale(100, 100);
addObject(coin2, 1400, 170); // Coin di atas Crate tinggi

// Set 3: Platform bertingkat dengan kombinasi Crate
PlatformTiles platform3 = new PlatformTiles();
addObject(platform3, 2000, 400); // Platform sedang

PlatformTiles platform4 = new PlatformTiles();
addObject(platform4, 2200, 340); // Platform lebih tinggi
```

```
Crate1 crate5 = new Crate1();
addObject(crate5, 2300, 407); // Crate rendah setelah platform

Coin coin3 = new Coin();
coin3.getImage().scale(100, 100);
addObject(coin3, 2250, 170); // Coin antara Platform dan Crate

Heart heart2 = new Heart();
heart2.getImage().scale(100, 100);
addObject(heart2, 2500, 200); // Heart kedua, perlu lompatan maksimal untuk dicapai

// Set 4: Platform panjang dan Crate bertingkat sulit
PlatformTiles platform5 = new PlatformTiles();
addObject(platform5, 3000, 380); // Platform tinggi

Crate1 crate6 = new Crate1();
addObject(crate6, 3200, 407); // Crate rendah

Crate1 crate7 = new Crate1();
addObject(crate7, 3350, 330); // Crate tinggi di ujung platform

Coin coin4 = new Coin();
coin4.getImage().scale(100, 100);
addObject(coin4, 3350, 130); // Coin di atas crate tinggi

// Set 5: Tantangan akhir dengan Platform panjang dan Heart terakhir
PlatformTiles platform6 = new PlatformTiles();
addObject(platform6, 4500, 360); // Platform sedang mendekati akhir

PlatformTiles platform7 = new PlatformTiles();
addObject(platform7, 4700, 290); // Platform tinggi mendekati akhir
```

```
heart3.getImage().scale(100, 100);
addObject(heart3, 4600, 200); // Heart ketiga, memerlukan lompatan

Crate1 crate8 = new Crate1();
addObject(crate8, 4900, 407); // Crate rendah mendekati akhir

Crate1 crate9 = new Crate1();
addObject(crate9, 5000, 330); // Crate bertingkat tinggi

Coin coin5 = new Coin();
coin5.getImage().scale(100, 100);
addObject(coin5, 5000, 130); // Coin di atas crate tinggi

// Set 6: Tantangan terakhir
PlatformTiles platform8 = new PlatformTiles();
addObject(platform8, 5500, 360); // Platform sedang mendekati akhir

Heart heart4 = new Heart();
heart4.getImage().scale(100, 100);
addObject(heart4, 5800, 200); // Heart terakhir sebelum akhir

Coin coin6 = new Coin();
coin6.getImage().scale(100, 100);
addObject(coin6, 5700, 120); // Coin di atas, perlu kombinasi lompatan

// Set 7: Platform bertingkat rendah dengan Crate dan Coin
PlatformTiles platform9 = new PlatformTiles();
addObject(platform9, 6200, 350); // Platform rendah

Crate1 crate10 = new Crate1();
addObject(crate10, 6400, 407); // Crate di tanah

Coin coin7 = new Coin();
coin7.getImage().scale(100, 100);
addObject(coin7, 6300, 170); // Coin di atas platform, masih dalam aturan

Heart heart5 = new Heart();
heart5.getImage().scale(100, 100);
addObject(heart5, 6500, 120); // Heart di posisi aman, memerlukan lompatan

// Set 8: Tantangan Crate bertingkat dengan Platform tinggi
Crate1 crate11 = new Crate1();
addObject(crate11, 6800, 407); // Crate di tanah

Crate1 crate12 = new Crate1();
addObject(crate12, 6800, 330); // Crate bertingkat

PlatformTiles platform10 = new PlatformTiles();
addObject(platform10, 7000, 400); // Platform lebih tinggi

Coin coin8 = new Coin();
coin8.getImage().scale(100, 100);
addObject(coin8, 6900, 150); // Coin di atas crate bertingkat

Heart heart6 = new Heart();
heart6.getImage().scale(100, 100);
addObject(heart6, 7200, 100); // Heart di atas platform, memerlukan lompatan kombinasi

// Set 9: Platform panjang dengan Coin sulit dijangkau
PlatformTiles platform11 = new PlatformTiles();
addObject(platform11, 7600, 360); // Platform panjang

PlatformTiles platform12 = new PlatformTiles();
addObject(platform12, 7800, 300); // Platform lebih tinggi
```

```
    Coin coin9 = new Coin();
    coin9.getImage().scale(100, 100);
    addObject(coin9, 7700, 170); // Coin di udara, memerlukan lompatan

    Crate1 crate13 = new Crate1();
    addObject(crate13, 8000, 407); // Crate di tanah

    Heart heart7 = new Heart();
    heart7.getImage().scale(100, 100);
    addObject(heart7, 8100, 150); // Heart di atas crate, dalam batas aturan

    // Set 10: Tantangan akhir dengan Crate dan Platform
    PlatformTiles platform13 = new PlatformTiles();
    addObject(platform13, 8600, 400); // Platform panjang rendah

    Crate1 crate14 = new Crate1();
    addObject(crate14, 8800, 407); // Crate di tanah

    Crate1 crate15 = new Crate1();
    addObject(crate15, 8800, 330); // Crate bertingkat

    Coin coin10 = new Coin();
    coin10.getImage().scale(100, 100);
    addObject(coin10, 8700, 170); // Coin di udara di atas crate

    Heart heart8 = new Heart();
    heart8.getImage().scale(100, 100);
    addObject(heart8, 9000, 100); // Heart terakhir sebelum akhir level
}
```

```
private void spawnInitialZombies() {
    zombie zombie1 = new zombie();
    addObject(zombie1, 900, 526);

    zombie zombie2 = new zombie();
    addObject(zombie2, 1200, 526);

    zombie zombie3 = new zombie();
    addObject(zombie3, 1500, 526);

    zombie zombie4 = new zombie();
    addObject(zombie4, 2600, 526);

    zombie zombie5 = new zombie();
    addObject(zombie5, 2000, 526);

    zombie zombie6 = new zombie();
    addObject(zombie6, 2300, 526);

    zombie zombie7 = new zombie();
    addObject(zombie7, 3500, 526);

    zombie zombie8 = new zombie();
    addObject(zombie8, 4000, 526);

    zombie zombie9 = new zombie();
    addObject(zombie9, 4800, 526);

    zombie zombie10 = new zombie();
    addObject(zombie10, 5400, 526);

    zombie zombie11 = new zombie();
    addObject(zombie11, 5900, 526);

    zombie zombie12 = new zombie();
    addObject(zombie12, 6800, 526);

    zombie zombie13 = new zombie();
    addObject(zombie13, 7400, 526);

    zombie zombie14 = new zombie();
    addObject(zombie14, 8000, 526);

    zombie zombie15 = new zombie();
    addObject(zombie15, 9000, 526);

}
```

```
private void spawnInitialSnakes() {
    snake snake1 = new snake();
    addObject(snake1,899,551);

    snake snake2 = new snake();
    addObject(snake2,1000,551);

    snake snake3 = new snake();
    addObject(snake3,1200,551);

    snake snake4 = new snake();
    addObject(snake4, 6200, 551);

    snake snake5 = new snake();
    addObject(snake5, 6600, 551);

    snake snake6 = new snake();
    addObject(snake6, 7900, 551);

    snake snake7 = new snake();
    addObject(snake7, 8600, 551);

    snake snake8 = new snake();
    addObject(snake8, 9100, 551);

    snake snake9 = new snake();
    addObject(snake9, 9500, 551);

    snake snake10 = new snake();
    addObject(snake10, 9800, 551);
}
```

```
private void createGroundTiles(int tileWidth, int groundY) {
    int numTiles = (int) Math.ceil((double) 10000 / tileWidth); // 4000 lebar dunia
    for (int i = 0; i < numTiles; i++) {
        Tiles tile = new Tiles();
        addObject(tile, i * tileWidth, groundY);
    }
}
```

```
private void spawnMultipleSnakes(int count) {
    List<Tiles> tiles = getObjects(Tiles.class);

    for (int i = 0; i < count; i++) {
        if (!tiles.isEmpty()) {
            int snakeX;
            int snakeY;
            Tiles randomTile;

            do {
                randomTile = tiles.get(Greenfoot.getRandomNumber(tiles.size()));
                snakeX = randomTile.getX();
            } while (snakeX <= 130);

            snakeY = randomTile.getY() - 100;

            snake snake = new snake();
            addObject(snake, snakeX, snakeY);
        }
    }
}
```

```java
private void addZombie(int x, int y) {
    zombie newZombie = new zombie();
    addObject(newZombie, x, y);
    zombieCount++;
}

private void spawnZombiesWithDelay() {
    int zombieX = 3980; // Posisi spawn zombie
    int groundY = 526;  // Posisi zombie di atas ground

    if (zombieCount < 12 && System.currentTimeMillis() - lastZombieSpawnTime >= zombieSpawnDelay) {
        addZombie(zombieX, groundY);
        lastZombieSpawnTime = System.currentTimeMillis();
    }
}
```

```java
/**
 * Logika kamera untuk mengikuti pemain di Level1.
 * Kamera bergerak mengikuti posisi pemain.
 */
private void followPlayerWithCamera() {
    if (getPlayer() == null) return;

    // Posisi pemain
    int playerX = getPlayer().getX();
    int playerY = getPlayer().getY();

    // posisi kamera, pemain mid
    int viewX = Math.max(viewWidth / 2, Math.min(playerX, worldWidth - viewWidth / 2));
    int offsetX = viewX - viewWidth / 2;

    // geser seluruh dunia berdasarkan offsetX
    for (Object obj : getObjects(null)) {
        Actor actor = (Actor) obj;
        int newX = actor.getX() - offsetX;
        actor.setLocation(newX, actor.getY());
    }

    // supaya ga keluar viewport
    int limitedPlayerX = Math.max(getPlayer().getImage().getWidth() / 2,
            Math.min(playerX, worldWidth - getPlayer().getImage().getWidth() / 2));
    if (playerX != limitedPlayerX) {
        getPlayer().setLocation(limitedPlayerX, playerY);
    }
}
```

```java
/**
 * Posisi tetap untuk skor.
 */
protected void keepCounterSticky() {
    if (getCounter() != null) {
        getCounter().setLocation(getCameraOffsetX() + 1000, 50);
    }
}

/**
 * Posisi tetap untuk health bar.
 */
protected void keepHealthBarSticky() {
    if (getHealthBar() != null) {
        getHealthBar().setLocation(getCameraOffsetX() + 180, 50);
    }
}

// Sticky for the timer
private void keepTimerSticky() {
    if (timer != null) {
        timer.setLocation(getCameraOffsetX() + 1180, 50);  // Keep the timer at the same position
    }
}
```

```java
    /**
     * Offset kamera untuk mengetahui posisi viewport.
     */
    private int getCameraOffsetX() {
        if (getPlayer() == null) return 0;

        int playerX = getPlayer().getX();
        return Math.max(0, Math.min(playerX - viewWidth / 2, worldWidth - viewWidth));
    }
}
```

## Level3

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;

public class Level2 extends BaseWorld {
    private int zombieCount = 0;            // Jumlah zombie di dunia
    private long lastZombieSpawnTime = 0;   // Waktu spawn zombie terakhir
    private int zombieSpawnDelay = 2000;    // Delay untuk spawn zombie (ms)

    private int viewWidth;  // Lebar viewport
    private int viewHeight; // Tinggi viewport
    private int worldWidth; // Lebar dunia

    private Timer timer;
    private GreenfootSound level2Song;
    public Level2() {
        super(1280, 720, 4000); // Viewport 1280x720, dunia 4000 px

        this.viewWidth = 1280;
        this.viewHeight = 720;
        this.worldWidth = 10000;

        setBackground(new GreenfootImage("BG2.png"));
        prepareLevel();

        level2Song = new GreenfootSound("Level2.mp3");
        level2Song.playLoop();
    }
```

```
    public void act() {
        // Logika kamera untuk mengikuti pemain
        followPlayerWithCamera(); // Fokus kamera pada pemain
        spawnZombiesWithDelay();  // Spawn zombie
        keepCounterSticky();      // Jaga posisi skor tetap
        keepHealthBarSticky();    // Jaga posisi health bar tetap
        keepTimerSticky();
    }

    @Override
    public void stopped() {
    level2Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level2Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level2Song.stop();
    }
```

```
private void prepareLevel() {
    // player
    Gatot_StandBy player = new Gatot_StandBy(90, 130);
    setPlayer(player, 100, 459); // Posisi awal pemain

    // tiles
    createGroundTiles(128, 651);

    // sign
    addObject(new Sign2(), 10000, 459); // Tanda akhir level

    // zombie
    spawnInitialZombies();

    // snake
    spawnInitialSnakes();

    // snake random
    spawnMultipleSnakes(10);

    //timer
    timer = new Timer();  // Initialize the timer
    addObject(timer, 1180, 50);  // Add the timer to the world

    // Set 1: Awal level - platform rendah dan Crate bertingkat
    PlatformTiles platform1 = new PlatformTiles();
    addObject(platform1, 400, 400); // Platform rendah dekat awal

    Crate1 crate1 = new Crate1();
    addObject(crate1, 600, 407); // Crate di tanah

    Crate1 crate2 = new Crate1();
    addObject(crate2, 600, 330); // Crate bertingkat untuk meloncat lebih tinggi

    Coin coin1 = new Coin();
    coin1.getImage().scale(100, 100);
    addObject(coin1, 600, 170); // Coin di atas Crate bertingkat

    Heart heart1 = new Heart();
    heart1.getImage().scale(100, 100);
    addObject(heart1, 800, 300); // Heart pertama di awal, mudah dicapai

    // Set 2: Gabungan Platform tinggi dan Crate bertingkat
    PlatformTiles platform2 = new PlatformTiles();
    addObject(platform2, 1200, 380); // Platform sedang

    Crate1 crate3 = new Crate1();
    addObject(crate3, 1400, 407); // Crate rendah

    Crate1 crate4 = new Crate1();
    addObject(crate4, 1400, 330); // Crate tinggi di atas crate rendah

    Coin coin2 = new Coin();
    coin2.getImage().scale(100, 100);
    addObject(coin2, 1400, 170); // Coin di atas Crate tinggi

    // Set 3: Platform bertingkat dengan kombinasi Crate
    PlatformTiles platform3 = new PlatformTiles();
    addObject(platform3, 2000, 400); // Platform sedang

    PlatformTiles platform4 = new PlatformTiles();
    addObject(platform4, 2200, 340); // Platform lebih tinggi
```

```
Crate1 crate5 = new Crate1();
addObject(crate5, 2300, 407); // Crate rendah setelah platform

Coin coin3 = new Coin();
coin3.getImage().scale(100, 100);
addObject(coin3, 2250, 170); // Coin antara Platform dan Crate

Heart heart2 = new Heart();
heart2.getImage().scale(100, 100);
addObject(heart2, 2500, 200); // Heart kedua, perlu lompatan maksimal untuk dicapai

// Set 4: Platform panjang dan Crate bertingkat sulit
PlatformTiles platform5 = new PlatformTiles();
addObject(platform5, 3000, 380); // Platform tinggi

Crate1 crate6 = new Crate1();
addObject(crate6, 3200, 407); // Crate rendah

Crate1 crate7 = new Crate1();
addObject(crate7, 3350, 330); // Crate tinggi di ujung platform

Coin coin4 = new Coin();
coin4.getImage().scale(100, 100);
addObject(coin4, 3350, 130); // Coin di atas crate tinggi

// Set 5: Tantangan akhir dengan Platform panjang dan Heart terakhir
PlatformTiles platform6 = new PlatformTiles();
addObject(platform6, 4500, 360); // Platform sedang mendekati akhir

PlatformTiles platform7 = new PlatformTiles();
addObject(platform7, 4700, 290); // Platform tinggi mendekati akhir
```

```
Heart heart3 = new Heart();
heart3.getImage().scale(100, 100);
addObject(heart3, 4600, 200); // Heart ketiga, memerlukan lompatan

Crate1 crate8 = new Crate1();
addObject(crate8, 4900, 407); // Crate rendah mendekati akhir

Crate1 crate9 = new Crate1();
addObject(crate9, 5000, 330); // Crate bertingkat tinggi

Coin coin5 = new Coin();
coin5.getImage().scale(100, 100);
addObject(coin5, 5000, 130); // Coin di atas crate tinggi

// Set 6: Tantangan terakhir
PlatformTiles platform8 = new PlatformTiles();
addObject(platform8, 5500, 360); // Platform sedang mendekati akhir

Heart heart4 = new Heart();
heart4.getImage().scale(100, 100);
addObject(heart4, 5800, 200); // Heart terakhir sebelum akhir

Coin coin6 = new Coin();
coin6.getImage().scale(100, 100);
addObject(coin6, 5700, 120); // Coin di atas, perlu kombinasi lompatan

// Set 7: Platform bertingkat rendah dengan Crate dan Coin
PlatformTiles platform9 = new PlatformTiles();
addObject(platform9, 6200, 350); // Platform rendah

Crate1 crate10 = new Crate1();
addObject(crate10, 6400, 407); // Crate di tanah
```

```
        Coin coin7 = new Coin();
        coin7.getImage().scale(100, 100);
        addObject(coin7, 6300, 170); // Coin di atas platform, masih dalam aturan

        Heart heart5 = new Heart();
        heart5.getImage().scale(100, 100);
        addObject(heart5, 6500, 120); // Heart di posisi aman, memerlukan lompatan

        // Set 8: Tantangan Crate bertingkat dengan Platform tinggi
        Crate1 crate11 = new Crate1();
        addObject(crate11, 6800, 407); // Crate di tanah

        Crate1 crate12 = new Crate1();
        addObject(crate12, 6800, 330); // Crate bertingkat

        PlatformTiles platform10 = new PlatformTiles();
        addObject(platform10, 7000, 400); // Platform lebih tinggi

        Coin coin8 = new Coin();
        coin8.getImage().scale(100, 100);
        addObject(coin8, 6900, 150); // Coin di atas crate bertingkat

        Heart heart6 = new Heart();
        heart6.getImage().scale(100, 100);
        addObject(heart6, 7200, 100); // Heart di atas platform, memerlukan lompatan kombinasi

        // Set 9: Platform panjang dengan Coin sulit dijangkau
        PlatformTiles platform11 = new PlatformTiles();
        addObject(platform11, 7600, 360); // Platform panjang
```

```
        PlatformTiles platform12 = new PlatformTiles();
        addObject(platform12, 7800, 300); // Platform lebih tinggi

        Coin coin9 = new Coin();
        coin9.getImage().scale(100, 100);
        addObject(coin9, 7700, 170); // Coin di udara, memerlukan lompatan

        Crate1 crate13 = new Crate1();
        addObject(crate13, 8000, 407); // Crate di tanah

        Heart heart7 = new Heart();
        heart7.getImage().scale(100, 100);
        addObject(heart7, 8100, 150); // Heart di atas crate, dalam batas aturan

        // Set 10: Tantangan akhir dengan Crate dan Platform
        PlatformTiles platform13 = new PlatformTiles();
        addObject(platform13, 8600, 400); // Platform panjang rendah

        Crate1 crate14 = new Crate1();
        addObject(crate14, 8800, 407); // Crate di tanah

        Crate1 crate15 = new Crate1();
        addObject(crate15, 8800, 330); // Crate bertingkat

        Coin coin10 = new Coin();
        coin10.getImage().scale(100, 100);
        addObject(coin10, 8700, 170); // Coin di udara di atas crate

        Heart heart8 = new Heart();
        heart8.getImage().scale(100, 100);
        addObject(heart8, 9000, 100); // Heart terakhir sebelum akhir level
    }
```

```
private void spawnInitialZombies() {
    zombie zombie1 = new zombie();
    addObject(zombie1, 900, 526);

    zombie zombie2 = new zombie();
    addObject(zombie2, 1200, 526);

    zombie zombie3 = new zombie();
    addObject(zombie3, 1500, 526);

    zombie zombie4 = new zombie();
    addObject(zombie4, 2600, 526);

    zombie zombie5 = new zombie();
    addObject(zombie5, 2000, 526);

    zombie zombie6 = new zombie();
    addObject(zombie6, 2300, 526);

    zombie zombie7 = new zombie();
    addObject(zombie7, 3500, 526);

    zombie zombie8 = new zombie();
    addObject(zombie8, 4000, 526);

    zombie zombie9 = new zombie();
    addObject(zombie9, 4800, 526);

    zombie zombie10 = new zombie();
    addObject(zombie10, 5400, 526);

    zombie zombie11 = new zombie();
    addObject(zombie11, 5900, 526);

    zombie zombie12 = new zombie();
    addObject(zombie12, 6800, 526);

    zombie zombie13 = new zombie();
    addObject(zombie13, 7400, 526);

    zombie zombie14 = new zombie();
    addObject(zombie14, 8000, 526);

    zombie zombie15 = new zombie();
    addObject(zombie15, 9000, 526);

}
```

```
private void spawnInitialSnakes() {
    snake snake1 = new snake();
    addObject(snake1,899,551);

    snake snake2 = new snake();
    addObject(snake2,1000,551);

    snake snake3 = new snake();
    addObject(snake3,1200,551);

    snake snake4 = new snake();
    addObject(snake4, 6200, 551);

    snake snake5 = new snake();
    addObject(snake5, 6600, 551);

    snake snake6 = new snake();
    addObject(snake6, 7900, 551);

    snake snake7 = new snake();
    addObject(snake7, 8600, 551);

    snake snake8 = new snake();
    addObject(snake8, 9100, 551);

    snake snake9 = new snake();
    addObject(snake9, 9500, 551);

    snake snake10 = new snake();
    addObject(snake10, 9800, 551);
}
```

```
private void createGroundTiles(int tileWidth, int groundY) {
    int numTiles = (int) Math.ceil((double) 10000 / tileWidth); // 4000 lebar dunia
    for (int i = 0; i < numTiles; i++) {
        Tiles tile = new Tiles();
        addObject(tile, i * tileWidth, groundY);
    }
}

private void spawnMultipleSnakes(int count) {
    List<Tiles> tiles = getObjects(Tiles.class);

    for (int i = 0; i < count; i++) {
        if (!tiles.isEmpty()) {
            int snakeX;
            int snakeY;
            Tiles randomTile;

            do {
                randomTile = tiles.get(Greenfoot.getRandomNumber(tiles.size()));
                snakeX = randomTile.getX();
            } while (snakeX <= 130);

            snakeY = randomTile.getY() - 100;

            snake snake = new snake();
            addObject(snake, snakeX, snakeY);
        }
    }
}
```

```java
private void addZombie(int x, int y) {
    zombie newZombie = new zombie();
    addObject(newZombie, x, y);
    zombieCount++;
}

private void spawnZombiesWithDelay() {
    int zombieX = 3980; // Posisi spawn zombie
    int groundY = 526;  // Posisi zombie di atas ground

    if (zombieCount < 12 && System.currentTimeMillis() - lastZombieSpawnTime >= zombieSpawnDelay) {
        addZombie(zombieX, groundY);
        lastZombieSpawnTime = System.currentTimeMillis();
    }
}
```

```java
/**
 * Logika kamera untuk mengikuti pemain di Level1.
 * Kamera bergerak mengikuti posisi pemain.
 */
private void followPlayerWithCamera() {
    if (getPlayer() == null) return;

    // Posisi pemain
    int playerX = getPlayer().getX();
    int playerY = getPlayer().getY();

    // posisi kamera, pemain mid
    int viewX = Math.max(viewWidth / 2, Math.min(playerX, worldWidth - viewWidth / 2));
    int offsetX = viewX - viewWidth / 2;

    // geser seluruh dunia berdasarkan offsetX
    for (Object obj : getObjects(null)) {
        Actor actor = (Actor) obj;
        int newX = actor.getX() - offsetX;
        actor.setLocation(newX, actor.getY());
    }

    // supaya ga keluar viewport
    int limitedPlayerX = Math.max(getPlayer().getImage().getWidth() / 2,
            Math.min(playerX, worldWidth - getPlayer().getImage().getWidth() / 2));
    if (playerX != limitedPlayerX) {
        getPlayer().setLocation(limitedPlayerX, playerY);
    }
}
```

```java
/**
 * Posisi tetap untuk skor.
 */
protected void keepCounterSticky() {
    if (getCounter() != null) {
        getCounter().setLocation(getCameraOffsetX() + 1000, 50);
    }
}

/**
 * Posisi tetap untuk health bar.
 */
protected void keepHealthBarSticky() {
    if (getHealthBar() != null) {
        getHealthBar().setLocation(getCameraOffsetX() + 180, 50);
    }
}

// Sticky for the timer
private void keepTimerSticky() {
    if (timer != null) {
        timer.setLocation(getCameraOffsetX() + 1180, 50);  // Keep the timer at the same position
    }
}
```

```
    /**
     * Offset kamera untuk mengetahui posisi viewport.
     */
    private int getCameraOffsetX() {
        if (getPlayer() == null) return 0;

        int playerX = getPlayer().getX();
        return Math.max(0, Math.min(playerX - viewWidth / 2, worldWidth - viewWidth));
    }
}
```

## Level4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;

public class Level4 extends BaseWorld {
    private int zombieCount = 0;            // Jumlah zombie di dunia
    private long lastZombieSpawnTime = 0;   // Waktu spawn zombie terakhir
    private int zombieSpawnDelay = 2000;    // Delay untuk spawn zombie (ms)

    private int viewWidth;  // Lebar viewport
    private int viewHeight; // Tinggi viewport
    private int worldWidth; // Lebar dunia

    private Timer timer;
    private GreenfootSound level4Song;
    public Level4() {
        super(1280, 720, 4000); // Viewport 1280x720, dunia 4000 px

        this.viewWidth = 1280;
        this.viewHeight = 720;
        this.worldWidth = 20000;

        setBackground(new GreenfootImage("BG4.png"));
        prepareLevel();

        level4Song = new GreenfootSound("Level4.mp3");
        level4Song.playLoop();
    }
```

```
public void act() {
    // Logika kamera untuk mengikuti pemain
    followPlayerWithCamera(); // Fokus kamera pada pemain
    spawnZombiesWithDelay();  // Spawn zombie
    keepCounterSticky();      // Jaga posisi skor tetap
    keepHealthBarSticky();    // Jaga posisi health bar tetap
    keepTimerSticky();
}

@Override
public void stopped() {
level4Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level4Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level4Song.stop();
}
```

```
private void prepareLevel() {
    // player
    Gatot_StandBy player = new Gatot_StandBy(90, 130);
    setPlayer(player, 100, 459); // Posisi awal pemain

    // tiles
    createGroundTiles(128, 651);

    // sign
    addObject(new Sign4(), 20000, 459); // Tanda akhir level

    // zombie
    spawnInitialZombies();

    // snake
    spawnInitialSnakes();

    // snake random
    spawnMultipleSnakes(20);

    //timer
    timer = new Timer();  // Initialize the timer
    addObject(timer, 1180, 50);  // Add the timer to the world

    // --- Set 1 ---
    PlatformTiles platform1 = new PlatformTiles();
    addObject(platform1, 400, 420);

    Crate1 crate1 = new Crate1();
    addObject(crate1, 500, 407);
```

```
    Coin coin1 = new Coin();
    coin1.getImage().scale(100, 100);
    addObject(coin1, 500, 300); // Coin di atas crate

    Heart heart1 = new Heart();
    heart1.getImage().scale(100, 100);
    addObject(heart1, 600, 280); // Heart di atas platform

    // --- Set 2 ---
    PlatformTiles platform2 = new PlatformTiles();
    addObject(platform2, 1000, 400);

    Crate1 crate2 = new Crate1();
    addObject(crate2, 1100, 407);

    Crate1 crate3 = new Crate1();
    addObject(crate3, 1100, 330);

    Coin coin2 = new Coin();
    coin2.getImage().scale(100, 100);
    addObject(coin2, 1150, 250); // Coin di atas crate

    // --- Set 3 ---
    PlatformTiles platform3 = new PlatformTiles();
    addObject(platform3, 1800, 360);

    PlatformTiles platform4 = new PlatformTiles();
    addObject(platform4, 2100, 300);

    Coin coin3 = new Coin();
    coin3.getImage().scale(100, 100);
    addObject(coin3, 1950, 200); // Coin di udara, lebih dekat platform
```

```
Heart heart2 = new Heart();
heart2.getImage().scale(100, 100);
addObject(heart2, 2200, 220); // Heart di atas platform

// --- Set 4 ---
Crate1 crate4 = new Crate1();
addObject(crate4, 2800, 407);

Coin coin4 = new Coin();
coin4.getImage().scale(100, 100);
addObject(coin4, 2800, 300); // Coin di atas crate

// --- Set 5 ---
PlatformTiles platform5 = new PlatformTiles();
addObject(platform5, 3600, 421);

Crate1 crate5 = new Crate1();
addObject(crate5, 3700, 407);

Crate1 crate6 = new Crate1();
addObject(crate6, 3700, 330);

Coin coin5 = new Coin();
coin5.getImage().scale(100, 100);
addObject(coin5, 3750, 250); // Coin di atas crate

// --- Set 6 ---
PlatformTiles platform6 = new PlatformTiles();
addObject(platform6, 4500, 380);

Coin coin6 = new Coin();
coin6.getImage().scale(100, 100);
addObject(coin6, 4550, 200); // Coin di atas platform

Crate1 crate7 = new Crate1();
addObject(crate7, 4700, 407);

// --- Set 7 ---
PlatformTiles platform7 = new PlatformTiles();
addObject(platform7, 5400, 400);

Crate1 crate8 = new Crate1();
addObject(crate8, 5500, 407);

Coin coin7 = new Coin();
coin7.getImage().scale(100, 100);
addObject(coin7, 5500, 300); // Coin di atas crate

// --- Set 8 ---
PlatformTiles platform8 = new PlatformTiles();
addObject(platform8, 6200, 360);

PlatformTiles platform9 = new PlatformTiles();
addObject(platform9, 6500, 300);

Coin coin8 = new Coin();
coin8.getImage().scale(100, 100);
addObject(coin8, 6400, 220); // Coin di udara

Heart heart3 = new Heart();
heart3.getImage().scale(100, 100);
addObject(heart3, 6700, 200); // Heart di atas platform
```

```
    // --- Set 9 ---
    Crate1 crate9_1 = new Crate1();
    addObject(crate9_1, 7300, 407);

    Crate1 crate9_2 = new Crate1();
    addObject(crate9_2, 7400, 330);

    Crate1 crate9_3 = new Crate1();
    addObject(crate9_3, 7500, 260);

    Coin coin9 = new Coin();
    coin9.getImage().scale(100, 100);
    addObject(coin9, 7550, 150); // Coin di atas crate

    // --- Set 10 ---
    PlatformTiles platform10 = new PlatformTiles();
    addObject(platform10, 8200, 421);

    Crate1 crate10 = new Crate1();
    addObject(crate10, 8300, 407);

    Crate1 crate11 = new Crate1();
    addObject(crate11, 8300, 330);

    Coin coin10 = new Coin();
    coin10.getImage().scale(100, 100);
    addObject(coin10, 8350, 250);

    // --- Set 11 ---
    PlatformTiles platform11 = new PlatformTiles();
    addObject(platform11, 9200, 360);
```

```
    Coin coin11 = new Coin();
    coin11.getImage().scale(100, 100);
    addObject(coin11, 9300, 220);

    Heart heart4 = new Heart();
    heart4.getImage().scale(100, 100);
    addObject(heart4, 9500, 200);
}
```

```
private void spawnInitialZombies() {
    zombie zombie1 = new zombie();
    addObject(zombie1, 900, 526);

    zombie zombie2 = new zombie();
    addObject(zombie2, 1200, 526);

    zombie zombie3 = new zombie();
    addObject(zombie3, 1500, 526);

    zombie zombie4 = new zombie();
    addObject(zombie4, 2600, 526);

    zombie zombie5 = new zombie();
    addObject(zombie5, 2000, 526);

    zombie zombie6 = new zombie();
    addObject(zombie6, 2300, 526);

    zombie zombie7 = new zombie();
    addObject(zombie7, 3500, 526);

    zombie zombie8 = new zombie();
    addObject(zombie8, 4000, 526);

    zombie zombie9 = new zombie();
    addObject(zombie9, 4800, 526);

    zombie zombie10 = new zombie();
    addObject(zombie10, 5400, 526);

    zombie zombie11 = new zombie();
    addObject(zombie11, 5900, 526);

    zombie zombie12 = new zombie();
    addObject(zombie12, 6800, 526);

    zombie zombie13 = new zombie();
    addObject(zombie13, 7400, 526);

    zombie zombie14 = new zombie();
    addObject(zombie14, 8000, 526);

    zombie zombie15 = new zombie();
    addObject(zombie15, 9000, 526);

    zombie zombie16 = new zombie();
    addObject(zombie16, 9400, 526);

    zombie zombie17 = new zombie();
    addObject(zombie17, 10700, 526);

    zombie zombie18 = new zombie();
    addObject(zombie18, 12000, 526);

    zombie zombie19 = new zombie();
    addObject(zombie19, 13000, 526);

    zombie zombie20 = new zombie();
    addObject(zombie20, 14000, 526);
```

```
        zombie zombie21 = new zombie();
        addObject(zombie21, 15000, 526);

        zombie zombie22 = new zombie();
        addObject(zombie22, 16000, 526);

        zombie zombie23 = new zombie();
        addObject(zombie23, 17000, 526);

        zombie zombie24 = new zombie();
        addObject(zombie24, 18000, 526);

        zombie zombie25 = new zombie();
        addObject(zombie25, 19000, 526);

    }
    private void spawnInitialSnakes() {
        snake snake1 = new snake();
        addObject(snake1,899,551);

        snake snake2 = new snake();
        addObject(snake2,1000,551);

        snake snake3 = new snake();
        addObject(snake3,1200,551);

        snake snake4 = new snake();
        addObject(snake4, 6200, 551);

        snake snake5 = new snake();
        addObject(snake5, 6600, 551);

        snake snake6 = new snake();
        addObject(snake6, 7900, 551);

        snake snake7 = new snake();
        addObject(snake7, 8600, 551);

        snake snake8 = new snake();
        addObject(snake8, 9100, 551);

        snake snake9 = new snake();
        addObject(snake9, 9500, 551);

        snake snake10 = new snake();
        addObject(snake10, 9800, 551);
```

```
    snake snake11 = new snake();
    addObject(snake11, 11000, 551);

    snake snake12 = new snake();
    addObject(snake12, 11500, 551);

    snake snake13 = new snake();
    addObject(snake13, 12700, 551);

    snake snake14 = new snake();
    addObject(snake14, 14600, 551);

    snake snake15 = new snake();
    addObject(snake15, 15800, 551);

    snake snake16 = new snake();
    addObject(snake16, 17000, 551);

    snake snake17 = new snake();
    addObject(snake17, 18200, 551);

    snake snake18 = new snake();
    addObject(snake18, 19100, 551);
}
```

```
private void createGroundTiles(int tileWidth, int groundY) {
    int numTiles = (int) Math.ceil((double) 20000 / tileWidth); // 4000 lebar dunia
    for (int i = 0; i < numTiles; i++) {
        Tiles2 tile = new Tiles2();
        addObject(tile, i * tileWidth, groundY);
    }
}
```

```
private void spawnMultipleSnakes(int count) {
    List<Tiles> tiles = getObjects(Tiles.class);

    for (int i = 0; i < count; i++) {
        if (!tiles.isEmpty()) {
            int snakeX;
            int snakeY;
            Tiles randomTile;

            do {
                randomTile = tiles.get(Greenfoot.getRandomNumber(tiles.size()));
                snakeX = randomTile.getX();
            } while (snakeX <= 130);

            snakeY = randomTile.getY() - 100;

            snake snake = new snake();
            addObject(snake, snakeX, snakeY);
        }
    }
}
```

```java
private void addZombie(int x, int y) {
    zombie newZombie = new zombie();
    addObject(newZombie, x, y);
    zombieCount++;
}

private void spawnZombiesWithDelay() {
    int zombieX = 3980; // Posisi spawn zombie
    int groundY = 526;  // Posisi zombie di atas ground

    if (zombieCount < 20 && System.currentTimeMillis() - lastZombieSpawnTime >= zombieSpawnDelay) {
        addZombie(zombieX, groundY);
        lastZombieSpawnTime = System.currentTimeMillis();
    }
}
```

```java
/**
 * Logika kamera untuk mengikuti pemain di Level1.
 * Kamera bergerak mengikuti posisi pemain.
 */
private void followPlayerWithCamera() {
    if (getPlayer() == null) return;

    // Posisi pemain
    int playerX = getPlayer().getX();
    int playerY = getPlayer().getY();

    // posisi kamera, pemain mid
    int viewX = Math.max(viewWidth / 2, Math.min(playerX, worldWidth - viewWidth / 2));
    int offsetX = viewX - viewWidth / 2;

    // geser seluruh dunia berdasarkan offsetX
    for (Object obj : getObjects(null)) {
        Actor actor = (Actor) obj;
        int newX = actor.getX() - offsetX;
        actor.setLocation(newX, actor.getY());
    }

    // supaya ga keluar viewport
    int limitedPlayerX = Math.max(getPlayer().getImage().getWidth() / 2,
            Math.min(playerX, worldWidth - getPlayer().getImage().getWidth() / 2));
    if (playerX != limitedPlayerX) {
        getPlayer().setLocation(limitedPlayerX, playerY);
    }
}
```

```
/**
 * Posisi tetap untuk skor.
 */
protected void keepCounterSticky() {
    if (getCounter() != null) {
        getCounter().setLocation(getCameraOffsetX() + 1000, 50);
    }
}

/**
 * Posisi tetap untuk health bar.
 */
protected void keepHealthBarSticky() {
    if (getHealthBar() != null) {
        getHealthBar().setLocation(getCameraOffsetX() + 180, 50);
    }
}

// Sticky for the timer
private void keepTimerSticky() {
    if (timer != null) {
        timer.setLocation(getCameraOffsetX() + 1180, 50);  // Keep the timer at the same position
    }
}
```

```
    /**
     * Offset kamera untuk mengetahui posisi viewport.
     */
    private int getCameraOffsetX() {
        if (getPlayer() == null) return 0;

        int playerX = getPlayer().getX();
        return Math.max(0, Math.min(playerX - viewWidth / 2, worldWidth - viewWidth));
    }
}
```

## Level5

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;

public class Level5 extends BaseWorld {
    private int zombieCount = 0;            // Jumlah zombie di dunia
    private long lastZombieSpawnTime = 0;   // Waktu spawn zombie terakhir
    private int zombieSpawnDelay = 2000;    // Delay untuk spawn zombie (ms)

    private int viewWidth;  // Lebar viewport
    private int viewHeight; // Tinggi viewport
    private int worldWidth; // Lebar dunia

    private Timer timer;
    private GreenfootSound level5Song;
    public Level5() {
        super(1280, 720, 4000); // Viewport 1280x720, dunia 4000 px

        this.viewWidth = 1280;
        this.viewHeight = 720;
        this.worldWidth = 25000;

        setBackground(new GreenfootImage("BG5.png"));
        prepareLevel();

        level5Song = new GreenfootSound("th06_06.mp3");
        level5Song.playLoop();
    }
```

```
public void act() {
    // Logika kamera untuk mengikuti pemain
    followPlayerWithCamera(); // Fokus kamera pada pemain
    spawnZombiesWithDelay();  // Spawn zombie
    keepCounterSticky();      // Jaga posisi skor tetap
    keepHealthBarSticky();    // Jaga posisi health bar tetap
    keepTimerSticky();
}

@Override
public void stopped() {
level5Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level5Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level5Song.stop();
}
```

```
private void prepareLevel() {
    // player
    Gatot_StandBy player = new Gatot_StandBy(90, 130);
    setPlayer(player, 100, 459); // Posisi awal pemain

    // tiles
    createGroundTiles(128, 651);

    // sign
    addObject(new Sign5(), 25000, 459); // Tanda akhir level

    // zombie
    spawnInitialZombies();

    // snake
    spawnInitialSnakes();

    // snake random
    spawnMultipleSnakes(25);

    //timer
    timer = new Timer();  // Initialize the timer
    addObject(timer, 1180, 50);  // Add the timer to the world

    // Set 1: Platform with Heart and Coin
    PlatformTiles platform1 = new PlatformTiles();
    addObject(platform1, 800, 400);
    Heart heart1 = new Heart();
    heart1.getImage().scale(100, 100);
    addObject(heart1, 850, 150);
    Coin coin1 = new Coin();
```

```
coin1.getImage().scale(100, 100);
addObject(coin1, 900, 170);

// Set 2: Two crates with a Coin on top
Crate1 crate2_1 = new Crate1();
addObject(crate2_1, 1400, 407);
Crate1 crate2_2 = new Crate1();
addObject(crate2_2, 1500, 330);
Coin coin2 = new Coin();
coin2.getImage().scale(100, 100);
addObject(coin2, 1500, 170);

// Set 3: Mixed platform and crate
PlatformTiles platform3 = new PlatformTiles();
addObject(platform3, 2100, 421);
Crate1 crate3 = new Crate1();
addObject(crate3, 2200, 330);
Heart heart3 = new Heart();
heart3.getImage().scale(100, 100);
addObject(heart3, 2200, 100);

// Set 4: Single crate with Coin
Crate1 crate4 = new Crate1();
addObject(crate4, 2800, 350);
Coin coin4 = new Coin();
coin4.getImage().scale(100, 100);
addObject(coin4, 2850, 150);
```

```
// Set 5: Two platforms with Coin and Heart
PlatformTiles platform5_1 = new PlatformTiles();
addObject(platform5_1, 3500, 421);
PlatformTiles platform5_2 = new PlatformTiles();
addObject(platform5_2, 3700, 350);
Coin coin5 = new Coin();
coin5.getImage().scale(100, 100);
addObject(coin5, 3650, 250);
Heart heart5 = new Heart();
heart5.getImage().scale(100, 100);
addObject(heart5, 3750, 200);

// Set 6: Crate staircase
Crate1 crate6_1 = new Crate1();
addObject(crate6_1, 4300, 407);
Crate1 crate6_2 = new Crate1();
addObject(crate6_2, 4400, 330);
Crate1 crate6_3 = new Crate1();
addObject(crate6_3, 4500, 250);
Coin coin6 = new Coin();
coin6.getImage().scale(100, 100);
addObject(coin6, 4550, 150);
```

```
// Set 7: Long platform with multiple coins
PlatformTiles platform7 = new PlatformTiles();
addObject(platform7, 5200, 400);
Crate1 crate100 = new Crate1();
addObject(crate100, 5000, 320);
for (int i = 0; i < 5; i++) {
    Coin coin7 = new Coin();
    coin7.getImage().scale(100, 100);
    addObject(coin7, 4800 + (i * 100), 90);
}

// Set 8: Platform and crate
PlatformTiles platform8 = new PlatformTiles();
addObject(platform8, 6000, 421);
Crate1 crate8 = new Crate1();
addObject(crate8, 6100, 330);
Coin coin8 = new Coin();
coin8.getImage().scale(100, 100);
addObject(coin8, 6150, 250);

// Set 9: Two platforms with heart and coin
PlatformTiles platform9_1 = new PlatformTiles();
addObject(platform9_1, 6800, 421);
PlatformTiles platform9_2 = new PlatformTiles();
addObject(platform9_2, 7000, 350);
Heart heart9 = new Heart();
heart9.getImage().scale(100, 100);
addObject(heart9, 7050, 270);
Coin coin9 = new Coin();
coin9.getImage().scale(100, 100);
addObject(coin9, 7100, 200);
```

```
// Set 10: Crate bridge
for (int i = 0; i < 5; i++) {
    Crate1 crate10 = new Crate1();
    addObject(crate10, 7700 + (i * 80), 407);
}

// Set 11: Platform with coin
PlatformTiles platform11 = new PlatformTiles();
addObject(platform11, 8400, 421);
Coin coin11 = new Coin();
coin11.getImage().scale(100, 100);
addObject(coin11, 8450, 370);

// Set 12: Crate staircase with heart
Crate1 crate12_1 = new Crate1();
addObject(crate12_1, 9200, 407);
Crate1 crate12_2 = new Crate1();
addObject(crate12_2, 9300, 330);
Crate1 crate12_3 = new Crate1();
addObject(crate12_3, 9400, 250);
Heart heart12 = new Heart();
heart12.getImage().scale(100, 100);
addObject(heart12, 9450, 150);
```

```
// Set 13: Two crates with Coin above the second
Crate1 crate13_1 = new Crate1();
addObject(crate13_1, 10000, 407);
Crate1 crate13_2 = new Crate1();
addObject(crate13_2, 10100, 330);
Coin coin13 = new Coin();
coin13.getImage().scale(100, 100);
addObject(coin13, 10100, 230);

// Set 14: Platform with Crate and Heart above
PlatformTiles platform14 = new PlatformTiles();
addObject(platform14, 10800, 421);
Crate1 crate14 = new Crate1();
addObject(crate14, 10900, 330);
Heart heart14 = new Heart();
heart14.getImage().scale(100, 100);
addObject(heart14, 10950, 250);

// Set 15: Long platform with Coins in a line
PlatformTiles platform15 = new PlatformTiles();
addObject(platform15, 11600, 421);
for (int i = 0; i < 4; i++) {
    Coin coin15 = new Coin();
    coin15.getImage().scale(100, 100);
    addObject(coin15, 11600 + (i * 120), 370);
}
```

```
// Set 16: Crate staircase with Heart at the top
Crate1 crate16_1 = new Crate1();
addObject(crate16_1, 12500, 407);
Crate1 crate16_2 = new Crate1();
addObject(crate16_2, 12600, 330);
Crate1 crate16_3 = new Crate1();
addObject(crate16_3, 12700, 250);
Heart heart16 = new Heart();
heart16.getImage().scale(100, 100);
addObject(heart16, 12750, 150);

// Set 17: Mixed platform and crate challenge
PlatformTiles platform17_1 = new PlatformTiles();
addObject(platform17_1, 13500, 421);
Crate1 crate17_1 = new Crate1();
addObject(crate17_1, 13600, 330);
PlatformTiles platform17_2 = new PlatformTiles();
addObject(platform17_2, 13700, 250);
Coin coin17 = new Coin();
coin17.getImage().scale(100, 100);
addObject(coin17, 13750, 200);

// Set 18: Platform with multiple crates underneath
PlatformTiles platform18 = new PlatformTiles();
addObject(platform18, 14500, 400);
Coin coin18 = new Coin();
coin18.getImage().scale(100, 100);
addObject(coin18, 14700, 250);
```

```
// Set 19: Heart and Coins on alternating platforms
PlatformTiles platform19_1 = new PlatformTiles();
addObject(platform19_1, 15500, 421);
PlatformTiles platform19_2 = new PlatformTiles();
addObject(platform19_2, 15700, 350);
PlatformTiles platform19_3 = new PlatformTiles();
addObject(platform19_3, 15900, 280);
Heart heart19 = new Heart();
heart19.getImage().scale(100, 100);
addObject(heart19, 15750, 250);
for (int i = 0; i < 2; i++) {
    Coin coin19 = new Coin();
    coin19.getImage().scale(100, 100);
    addObject(coin19, 15550 + (i * 400), 370 - (i * 70));
}

// Set 20: Crate staircase with Heart at the top
Crate1 crate20_1 = new Crate1();
addObject(crate20_1, 16500, 407);
Crate1 crate20_2 = new Crate1();
addObject(crate20_2, 16600, 330);
Crate1 crate20_3 = new Crate1();
addObject(crate20_3, 16700, 250);
Heart heart20 = new Heart();
heart20.getImage().scale(100, 100);
addObject(heart20, 16750, 150);
```

```
// Set 21: Multi-platform with coins at different heights
PlatformTiles platform21_1 = new PlatformTiles();
addObject(platform21_1, 17000, 421);
PlatformTiles platform21_2 = new PlatformTiles();
addObject(platform21_2, 17200, 350);
PlatformTiles platform21_3 = new PlatformTiles();
addObject(platform21_3, 17400, 280);
for (int i = 0; i < 3; i++) {
    Coin coin21 = new Coin();
    coin21.getImage().scale(100, 100);
    addObject(coin21, 17050 + (i * 200), 250 - (i * 50));
}
Heart heart21 = new Heart();
heart21.getImage().scale(100, 100);
addObject(heart21, 17450, 200);

// Set 22: Crates in a triangular formation with heart at the peak
Crate1 crate22_1 = new Crate1();
addObject(crate22_1, 18000, 407);
Crate1 crate22_2 = new Crate1();
addObject(crate22_2, 18100, 330);
Crate1 crate22_3 = new Crate1();
addObject(crate22_3, 18200, 250);
Crate1 crate22_4 = new Crate1();
addObject(crate22_4, 18300, 330);
Crate1 crate22_5 = new Crate1();
addObject(crate22_5, 18400, 407);
Heart heart22 = new Heart();
heart22.getImage().scale(100, 100);
addObject(heart22, 18200, 150);
```

```
        // Set 23: Alternating platforms with koin at high levels
        PlatformTiles platform23_1 = new PlatformTiles();
        addObject(platform23_1, 19000, 421);
        PlatformTiles platform23_2 = new PlatformTiles();
        addObject(platform23_2, 19200, 320);
        PlatformTiles platform23_3 = new PlatformTiles();
        addObject(platform23_3, 19400, 250);
        for (int i = 0; i < 3; i++) {
            Coin coin23 = new Coin();
            coin23.getImage().scale(100, 100);
            addObject(coin23, 19050 + (i * 200), 200 - (i * 50));
        }
        Heart heart23 = new Heart();
        heart23.getImage().scale(100, 100);
        addObject(heart23, 19450, 180);

        // Set 24: Final challenge, high staircase with heart at the top
        Crate1 crate24_1 = new Crate1();
        addObject(crate24_1, 20000, 407);
        Crate1 crate24_2 = new Crate1();
        addObject(crate24_2, 20100, 330);
        Crate1 crate24_3 = new Crate1();
        addObject(crate24_3, 20200, 250);
        Crate1 crate24_4 = new Crate1();
        addObject(crate24_4, 20300, 170);
        Heart heart24 = new Heart();
        heart24.getImage().scale(100, 100);
        addObject(heart24, 20350, 100);


        PlatformTiles platform25 = new PlatformTiles();
        addObject(platform25, 22500, 421);
        Crate1 crate25 = new Crate1();
        addObject(crate25, 22600, 330);
        Coin coin25 = new Coin();
        coin25.getImage().scale(100, 100);
        addObject(coin25, 22700, 250);
        Heart heart25 = new Heart();
        heart25.getImage().scale(100, 100);
        addObject(heart25, 22800, 150);
    }
```

```
    private void spawnInitialZombies() {
        zombie zombie1 = new zombie();
        addObject(zombie1, 900, 526);

        zombie zombie2 = new zombie();
        addObject(zombie2, 1200, 526);

        zombie zombie3 = new zombie();
        addObject(zombie3, 1500, 526);

        zombie zombie4 = new zombie();
        addObject(zombie4, 2600, 526);

        zombie zombie5 = new zombie();
        addObject(zombie5, 2000, 526);

        zombie zombie6 = new zombie();
        addObject(zombie6, 2300, 526);

        zombie zombie7 = new zombie();
        addObject(zombie7, 3500, 526);

        zombie zombie8 = new zombie();
        addObject(zombie8, 4000, 526);

        zombie zombie9 = new zombie();
        addObject(zombie9, 4800, 526);

        zombie zombie10 = new zombie();
        addObject(zombie10, 5400, 526);
```

```
        zombie zombie11 = new zombie();
        addObject(zombie11, 5900, 526);

        zombie zombie12 = new zombie();
        addObject(zombie12, 6800, 526);

        zombie zombie13 = new zombie();
        addObject(zombie13, 7400, 526);

        zombie zombie14 = new zombie();
        addObject(zombie14, 8000, 526);

        zombie zombie15 = new zombie();
        addObject(zombie15, 9000, 526);

        zombie zombie16 = new zombie();
        addObject(zombie16, 9400, 526);

        zombie zombie17 = new zombie();
        addObject(zombie17, 10700, 526);

        zombie zombie18 = new zombie();
        addObject(zombie18, 12000, 526);

        zombie zombie19 = new zombie();
        addObject(zombie19, 13000, 526);

        zombie zombie20 = new zombie();
        addObject(zombie20, 14000, 526);
```

```
        zombie zombie21 = new zombie();
        addObject(zombie21, 15000, 526);

        zombie zombie22 = new zombie();
        addObject(zombie22, 16000, 526);

        zombie zombie23 = new zombie();
        addObject(zombie23, 17000, 526);

        zombie zombie24 = new zombie();
        addObject(zombie24, 18000, 526);

        zombie zombie25 = new zombie();
        addObject(zombie25, 19000, 526);

        zombie zombie26 = new zombie();
        addObject(zombie26, 20000, 526);

        zombie zombie27 = new zombie();
        addObject(zombie27, 21000, 526);

        zombie zombie28 = new zombie();
        addObject(zombie28, 22000, 526);

        zombie zombie29 = new zombie();
        addObject(zombie29, 23000, 526);

        zombie zombie30 = new zombie();
        addObject(zombie30, 24000, 526);
    }
```

```
private void spawnInitialSnakes() {
    snake snake1 = new snake();
    addObject(snake1,899,551);

    snake snake2 = new snake();
    addObject(snake2,1000,551);

    snake snake3 = new snake();
    addObject(snake3,1200,551);

    snake snake4 = new snake();
    addObject(snake4, 6200, 551);

    snake snake5 = new snake();
    addObject(snake5, 6600, 551);

    snake snake6 = new snake();
    addObject(snake6, 7900, 551);

    snake snake7 = new snake();
    addObject(snake7, 8600, 551);

    snake snake8 = new snake();
    addObject(snake8, 9100, 551);

    snake snake9 = new snake();
    addObject(snake9, 9500, 551);

    snake snake10 = new snake();
    addObject(snake10, 9800, 551);

    snake snake11 = new snake();
    addObject(snake11, 11000, 551);

    snake snake12 = new snake();
    addObject(snake12, 11500, 551);

    snake snake13 = new snake();
    addObject(snake13, 12700, 551);

    snake snake14 = new snake();
    addObject(snake14, 14600, 551);

    snake snake15 = new snake();
    addObject(snake15, 15800, 551);

    snake snake16 = new snake();
    addObject(snake16, 17000, 551);

    snake snake17 = new snake();
    addObject(snake17, 18200, 551);

    snake snake18 = new snake();
    addObject(snake18, 19100, 551);

    snake snake19 = new snake();
    addObject(snake19, 20000, 551);

    snake snake20 = new snake();
    addObject(snake20, 21200, 551);
```

```
        snake snake21 = new snake();
        addObject(snake21, 22400, 551);

        snake snake22 = new snake();
        addObject(snake22, 23500, 551);

        snake snake23 = new snake();
        addObject(snake23, 24200, 551);

        snake snake24 = new snake();
        addObject(snake24, 24700, 551);

        snake snake25 = new snake();
        addObject(snake25, 24900, 551);
    }
```

```
    private void createGroundTiles(int tileWidth, int groundY) {
        int numTiles = (int) Math.ceil((double) 25000 / tileWidth); // 4000 lebar dunia
        for (int i = 0; i < numTiles; i++) {
            Tiles2 tile = new Tiles2();
            addObject(tile, i * tileWidth, groundY);
        }
    }

    private void spawnMultipleSnakes(int count) {
        List<Tiles> tiles = getObjects(Tiles.class);

        for (int i = 0; i < count; i++) {
            if (!tiles.isEmpty()) {
                int snakeX;
                int snakeY;
                Tiles randomTile;

                do {
                    randomTile = tiles.get(Greenfoot.getRandomNumber(tiles.size()));
                    snakeX = randomTile.getX();
                } while (snakeX <= 130);

                snakeY = randomTile.getY() - 100;

                snake snake = new snake();
                addObject(snake, snakeX, snakeY);
            }
        }
    }
```

```
    private void addZombie(int x, int y) {
        zombie newZombie = new zombie();
        addObject(newZombie, x, y);
        zombieCount++;
    }

    private void spawnZombiesWithDelay() {
        int zombieX = 3980; // Posisi spawn zombie
        int groundY = 526;  // Posisi zombie di atas ground

        if (zombieCount < 24 && System.currentTimeMillis() - lastZombieSpawnTime >= zombieSpawnDelay) {
            addZombie(zombieX, groundY);
            lastZombieSpawnTime = System.currentTimeMillis();
        }
    }
```

```java
/**
 * Logika kamera untuk mengikuti pemain di Level1.
 * Kamera bergerak mengikuti posisi pemain.
 */
private void followPlayerWithCamera() {
    if (getPlayer() == null) return;

    // Posisi pemain
    int playerX = getPlayer().getX();
    int playerY = getPlayer().getY();

    // posisi kamera, pemain mid
    int viewX = Math.max(viewWidth / 2, Math.min(playerX, worldWidth - viewWidth / 2));
    int offsetX = viewX - viewWidth / 2;

    // geser seluruh dunia berdasarkan offsetX
    for (Object obj : getObjects(null)) {
        Actor actor = (Actor) obj;
        int newX = actor.getX() - offsetX;
        actor.setLocation(newX, actor.getY());
    }

    // supaya ga keluar viewport
    int limitedPlayerX = Math.max(getPlayer().getImage().getWidth() / 2,
            Math.min(playerX, worldWidth - getPlayer().getImage().getWidth() / 2));
    if (playerX != limitedPlayerX) {
        getPlayer().setLocation(limitedPlayerX, playerY);
    }
}
```

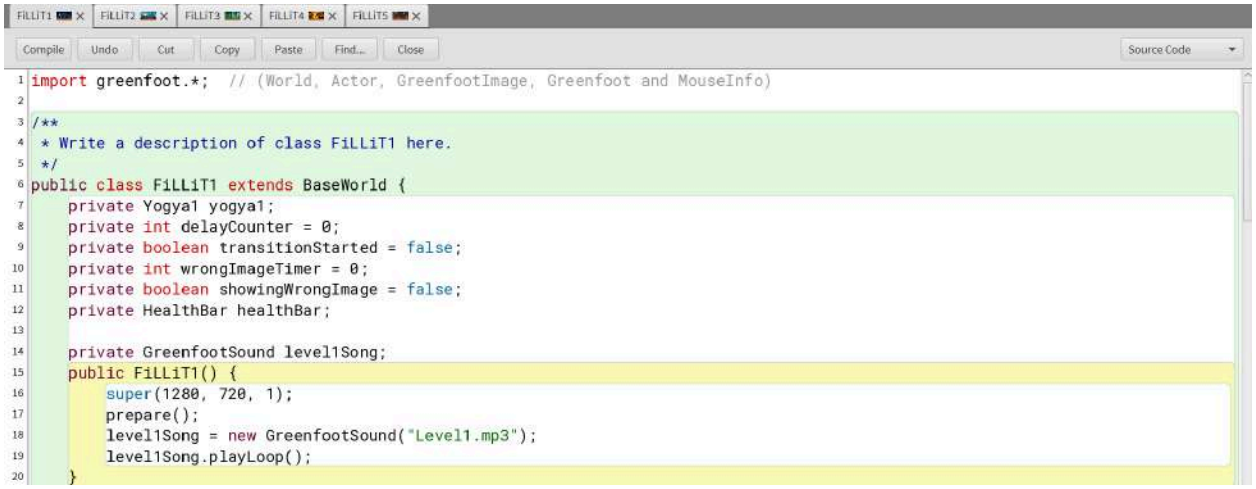
```java
/**
 * Posisi tetap untuk skor.
 */
protected void keepCounterSticky() {
    if (getCounter() != null) {
        getCounter().setLocation(getCameraOffsetX() + 1000, 50);
    }
}

/**
 * Posisi tetap untuk health bar.
 */
protected void keepHealthBarSticky() {
    if (getHealthBar() != null) {
        getHealthBar().setLocation(getCameraOffsetX() + 180, 50);
    }
}

// Sticky for the timer
private void keepTimerSticky() {
    if (timer != null) {
        timer.setLocation(getCameraOffsetX() + 1180, 50);  // Keep the timer at the same position
    }
}
```

```java
/**
 * Offset kamera untuk mengetahui posisi viewport.
 */
private int getCameraOffsetX() {
    if (getPlayer() == null) return 0;

    int playerX = getPlayer().getX();
    return Math.max(0, Math.min(playerX - viewWidth / 2, worldWidth - viewWidth));
}
}
```

Setelah menyelesaikan permainan pada world “Level” maka player akan berpindah ke world puzzle “FiLLiT”.

**World dimana player mengisi huruf yang kosong pada suatu kata terdiri dari FiLLiT1, FiLLiT2, FiLLiT3, FiLLiT4, FiLLiT5.**

**FiLLiT1**



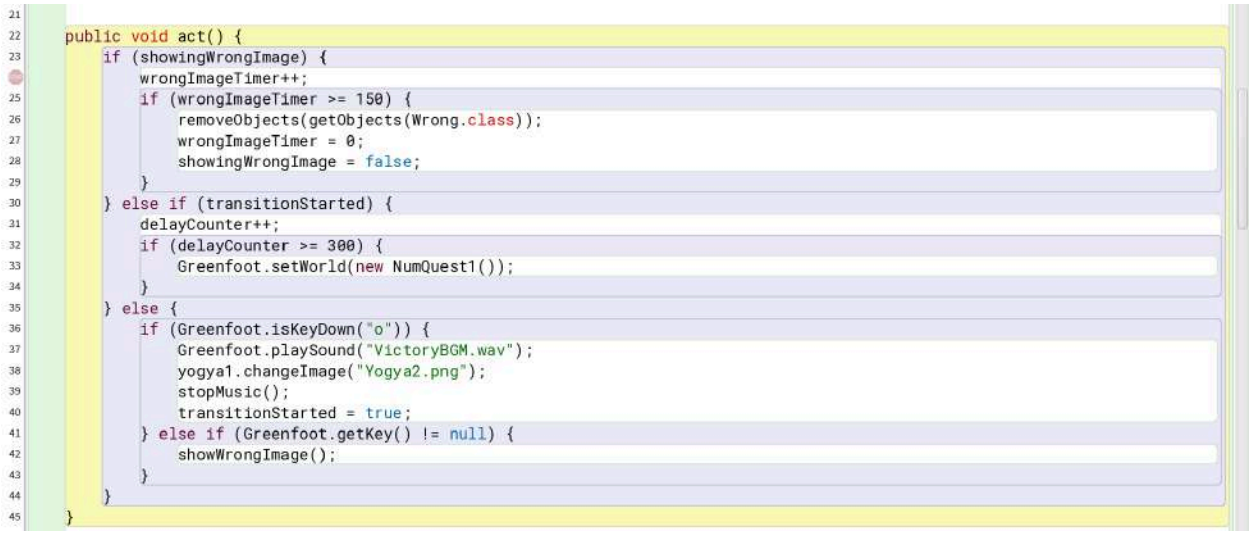

Penjelasan :

```
private Yogya1 yogya1;
private int delayCounter = 0;
private boolean transitionStarted = false;
private int wrongImageTimer = 0;
private boolean showingWrongImage = false;
private HealthBar healthBar;

private GreenfootSound level1Song;
```

> yogya1: Objek gambar jawaban yang masih kosong (perlu menerima input dari pemain untuk memberikan respon yang sesuai.
> delayCounter: Menghitung waktu delay untuk transisi ke world berikutnya.
> transitionStarted: Menandakan apakah transisi sudah dimulai.
> wrongImageTimer: Menghitung durasi tampilan Wrong Image.
> showingWrongImage: Menandakan apakah Wrong Image sedang ditampilkan.
> healthBar: Objek untuk menunjukkan health pemain.
> level1Song: Musik latar yang diputar selama level ini.

```
public FiLLiT1() {
    super(1280, 720, 1);
    prepare();
    level1Song = new GreenfootSound("Level1.mp3");
    level1Song.playLoop();
}
```

> super(1280, 720, 1): ukuran viewport world 1280x720 piksel, skala 1.
> Memanggil prepare() untuk menambahkan objek ke world.
> level1Song diatur untuk memutar musik latar dengan file Level1.mp3 dalam loop.

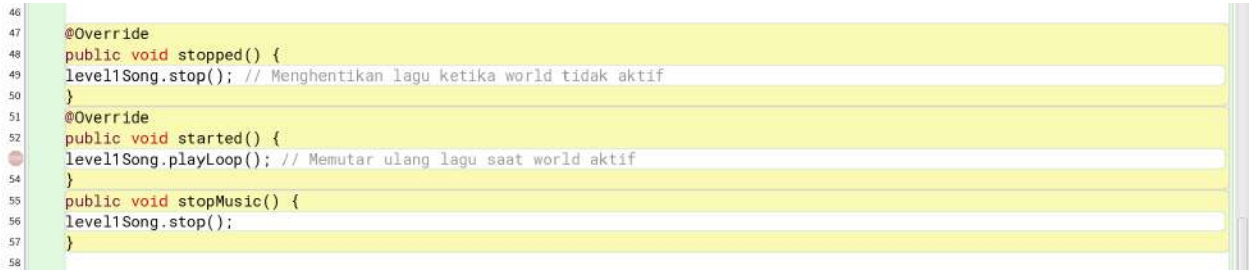


```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 150) {
            removeObjects(getObjects(Wrong.class));
            wrongImageTimer = 0;
            showingWrongImage = false;
        }
    } else if (transitionStarted) {
        delayCounter++;
        if (delayCounter >= 300) {
            Greenfoot.setWorld(new NumQuest1());
        }
    } else {
        if (Greenfoot.isKeyDown("o")) {
            Greenfoot.playSound("VictoryBGM.wav");
            yogya1.changeImage("Yogya2.png");
            stopMusic();
            transitionStarted = true;
        } else if (Greenfoot.getKey() != null) {
            showWrongImage();
        }
    }
}
```

Penjelasan :

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 150) {
            removeObjects(getObjects(Wrong.class));
            wrongImageTimer = 0;
            showingWrongImage = false;
        }
    } else if (transitionStarted) {
        delayCounter++;
        if (delayCounter >= 300) {
            Greenfoot.setWorld(new NumQuest1());
        }
    } else {
        if (Greenfoot.isKeyDown("o")) {
            Greenfoot.playSound("VictoryBGM.wav");
            yogya1.changeImage("Yogya2.png");
            stopMusic();
            transitionStarted = true;
        } else if (Greenfoot.getKey() != null) {
            showWrongImage();
        }
    }
```

```
}
```
> showingWrongImage: Jika gambar Wrong sedang ditampilkan, timer bertambah. Setelah 150 tick (2,5 detik), objek Wrong dihapus, timer direset, dan mode showingWrongImage dimatikan.
NOTE : default Greenfoot ialah 60 ticks per detik (60 FPS)
> transitionStarted: Jika transisi dimulai, delayCounter bertambah. Setelah 300 tick, world diganti menjadi NumQuest1.
> Input dari pemain: Tombol o: Mengganti gambar Yogya1 menjadi Yogya2, menghentikan musik, memutar musik VictoryBGM.wav sekali dan memulai transisi. Else, apabila player menekan tombol lain: menampilkan Wrong Image.



```
@Override
public void stopped() {
level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level1Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level1Song.stop();
}
```

Penjelasan :

```
@Override
public void stopped() {
level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level1Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level1Song.stop();
}
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

```
    private void prepare() {
        Wayang1 wayang1 = new Wayang1(500, 500);
        addObject(wayang1, 938, 372);
        wayang1.setLocation(1026, 368);

        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
        addObject(gatot_FiLLiT, 178, 399);
        gatot_FiLLiT.setLocation(200, 520);

        FiLLiT_1 fiLLiT_1 = new FiLLiT_1(300, 300);
        addObject(fiLLiT_1, 556, 344);
        fiLLiT_1.setLocation(722, 178);

        yogya1 = new Yogya1();
        addObject(yogya1, 421, 373);
```

```
        // Inisialisasi health bar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();
        addObject(timer, 1180, 50);
    }

    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;

        // Kurangi nyawa
        if (healthBar != null) {
            healthBar.decreaseLife(); // Kurangi health -1
        }
    }
}
```

Penjelasan :

```
private void prepareLevel() {
        …..;
}
```

> Method prepare() menempatkan objek sesuai dengan koordinat yang dibuat.

NOTE : Looping dilakukan untuk menambahkan tile setiap 128 px hingga lebar world tercapai. Tile ini menjadi dasar (ground) di posisi y = 651.

```
private void showWrongImage() {
    Wrong wrong = new Wrong();
    addObject(wrong, getWidth() / 2, getHeight() / 2);
    showingWrongImage = true;
    wrongImageTimer = 0;

    // Kurangi nyawa
    if (healthBar != null) {
        healthBar.decreaseLife(); // Kurangi health -1
```

```
    }
  }
```

> Menampilkan gambar Wrong di tengah layar.
> Mengaktifkan mode showingWrongImage dan mereset wrongImageTimer.
> Mengurangi nyawa pemain melalui healthBar.decreaseLife() jika health bar tersedia (jika gambar wrong ditampilkan maka health berkurang).

**FiLLiT2, FiLLiT3, FiLLiT4, FiLLiT5 memiliki kode Java yang sama seperti FiLLiT1, perbedaannya hanya terdapat pada :**

1. Musik
   - FiLLiT2 : level2Song = new GreenfootSound("Level2.mp3");
   - FiLLiT3 : level3Song = new GreenfootSound("Level3.mp3");
   - FiLLiT4 : level4Song = new GreenfootSound("Level4.mp3");
   - FiLLiT5 : level5Song = new GreenfootSound("th06_06.mp3");
2. Objek untuk setiap soal puzzle
   > Soal pada setiap world FiLLiT berbeda-beda, oleh karena itu objek gambar yang ditempatkan juga berbeda pada setiap world.
3. Input keyboard yang menyatakan bahwa jawaban benar untuk pindah ke world selanjutnya
   - FiLLiT2 : b
   - FiLLiT3 : b
   - FiLLiT4 : a
   - FiLLiT5 : t
4. Background (Background diset langsung melalui setimage pada setiap world FiLLiT)

**FiLLiT 2**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class FiLLiT1 here.
 */
public class FiLLiT2 extends BaseWorld {
    private Borobudur1 Borobudur1;
    private int delayCounter = 0;
    private boolean transitionStarted = false;
    private int wrongImageTimer = 0;
    private boolean showingWrongImage = false;
    private HealthBar healthBar; // Health bar untuk nyawa

    private GreenfootSound level2Song;
    public FiLLiT2() {
        super(1280, 720, 1);
        prepare();

        level2Song = new GreenfootSound("Level2.mp3");
        level2Song.playLoop();
    }
```

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 150) {
            removeObjects(getObjects(Wrong.class));
            wrongImageTimer = 0;
            showingWrongImage = false;
        }
    } else if (transitionStarted) {
        delayCounter++;
        if (delayCounter >= 300) {
            Greenfoot.setWorld(new NumQuest2());
        }
    } else {
        if (Greenfoot.isKeyDown("b")) {
            Greenfoot.playSound("VictoryBGM.wav");
            Borobudur1.changeImage("Borobudur2.png");
            stopMusic();
            transitionStarted = true;
        } else if (Greenfoot.getKey() != null) {
            showWrongImage();
        }
    }
}
```

```
@Override
public void stopped() {
level2Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level2Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level2Song.stop();
}
```

```
private void prepare() {
    Wayang1 wayang1 = new Wayang1(500, 500);
    addObject(wayang1, 938, 372);
    wayang1.setLocation(1026, 368);

    int worldWidth = getWidth();
    int tileWidth = 128;
    int yPosition = 651;

    for (int x = 0; x < worldWidth; x += tileWidth) {
        Tiles tile = new Tiles();
        addObject(tile, x + tileWidth / 2, yPosition);
    }

    Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
    addObject(gatot_FiLLiT, 178, 399);
    gatot_FiLLiT.setLocation(200, 520);

    Borobudur1 = new Borobudur1();
    addObject(Borobudur1, 421, 373);

    // Inisialisasi health bar
    healthBar = getHealthBar();

    //timer
    Timer timer = new Timer();  // Initialize the timer
    addObject(timer, 1180, 50);  // Add the timer to the world

    FiLLiT_2 fiLLiT_2 = new FiLLiT_2(300, 300);
    addObject(fiLLiT_2,722,178);

}
```

```
    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;

        // Kurangi nyawa
        if (healthBar != null) {
            healthBar.decreaseLife(); // Kurangi health -1
        }
    }
}
```

**FiLLiT 3**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class FiLLiT1 here.
 */
public class FiLLiT3 extends BaseWorld {
    private Rambutan1 Rambutan1;
    private int delayCounter = 0;
    private boolean transitionStarted = false;
    private int wrongImageTimer = 0;
    private boolean showingWrongImage = false;
    private HealthBar healthBar; // Health bar untuk nyawa

    private GreenfootSound level3Song;
    public FiLLiT3() {
        super(1280, 720, 1);
        prepare();
        level3Song = new GreenfootSound("Level3.mp3");
        level3Song.playLoop();
    }
```

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 150) {
            removeObjects(getObjects(Wrong.class));
            wrongImageTimer = 0;
            showingWrongImage = false;
        }
    } else if (transitionStarted) {
        delayCounter++;
        if (delayCounter >= 300) {
            Greenfoot.setWorld(new NumQuest3());
        }
    } else {
        if (Greenfoot.isKeyDown("b")) {
            Greenfoot.playSound("VictoryBGM.wav");
            Rambutan1.changeImage("Rambutan2.png");
            stopMusic();
            transitionStarted = true;
        } else if (Greenfoot.getKey() != null) {
            showWrongImage();
        }
    }
}
```

```
@Override
public void stopped() {
level3Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level3Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level3Song.stop();
}
```

```
    private void prepare() {
        Wayang1 wayang1 = new Wayang1(500, 500);
        addObject(wayang1, 938, 372);
        wayang1.setLocation(1026, 368);

        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
        addObject(gatot_FiLLiT, 178, 399);
        gatot_FiLLiT.setLocation(200, 520);

        Rambutan1 = new Rambutan1();
        addObject(Rambutan1, 421, 373);

        // Inisialisasi health bar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        FiLLiT_3 fiLLiT_3 = new FiLLiT_3(300, 300);
        addObject(fiLLiT_3,722,178);

    }
```

```
    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;

        // Kurangi nyawa
        if (healthBar != null) {
            healthBar.decreaseLife(); // Kurangi health -1
        }
    }
}
```

## FiLLiT 4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class FiLLiT1 here.
 */
public class FiLLiT4 extends BaseWorld {
    private Gatot1 Gatot1;
    private int delayCounter = 0;
    private boolean transitionStarted = false;
    private int wrongImageTimer = 0;
    private boolean showingWrongImage = false;
    private HealthBar healthBar; // Health bar untuk nyawa

    private GreenfootSound level4Song;
    public FiLLiT4() {
        super(1280, 720, 1);
        prepare();
        level4Song = new GreenfootSound("Level4.mp3");
        level4Song.playLoop();
    }
```

```
    public void act() {
        if (showingWrongImage) {
            wrongImageTimer++;
            if (wrongImageTimer >= 150) {
                removeObjects(getObjects(Wrong.class));
                wrongImageTimer = 0;
                showingWrongImage = false;
            }
        } else if (transitionStarted) {
            delayCounter++;
            if (delayCounter >= 300) {
                Greenfoot.setWorld(new NumQuest4());
            }
        } else {
            if (Greenfoot.isKeyDown("a")) {
                Greenfoot.playSound("VictoryBGM.wav");
                Gatot1.changeImage("Gatot2.png");
                stopMusic();
                transitionStarted = true;
            } else if (Greenfoot.getKey() != null) {
                showWrongImage();
            }
        }
    }
```

```
    @Override
    public void stopped() {
    level4Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level4Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level4Song.stop();
    }
```

```
    private void prepare() {
        Wayang1 wayang1 = new Wayang1(500, 500);
        addObject(wayang1, 938, 372);
        wayang1.setLocation(1026, 368);

        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
        addObject(gatot_FiLLiT, 178, 399);
        gatot_FiLLiT.setLocation(200, 520);

        Gatot1 = new Gatot1();
        addObject(Gatot1, 421, 373);

        // Inisialisasi health bar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        FiLLiT_4 fiLLiT_4 = new FiLLiT_4(300, 300);
        addObject(fiLLiT_4,722,178);

    }
```

```
    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;

        // Kurangi nyawa
        if (healthBar != null) {
            healthBar.decreaseLife(); // Kurangi health -1
        }
    }
}
```

## FiLLiT 5

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class FiLLiT1 here.
 */
public class FiLLiT5 extends BaseWorld {
    private Kartini1 Kartini1;
    private int delayCounter = 0;
    private boolean transitionStarted = false;
    private int wrongImageTimer = 0;
    private boolean showingWrongImage = false;
    private HealthBar healthBar; // Health bar untuk nyawa

    private GreenfootSound level5Song;
    public FiLLiT5() {
        super(1280, 720, 1);
        prepare();
        level5Song = new GreenfootSound("th06_06.mp3");
        level5Song.playLoop();
    }
```

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 150) {
            removeObjects(getObjects(Wrong.class));
            wrongImageTimer = 0;
            showingWrongImage = false;
        }
    } else if (transitionStarted) {
        delayCounter++;
        if (delayCounter >= 300) {
            Greenfoot.setWorld(new NumQuest5());
        }
    } else {
        if (Greenfoot.isKeyDown("t")) {
            Greenfoot.playSound("VictoryBGM.wav");
            Kartini1.changeImage("Kartini2.png");
            stopMusic();
            transitionStarted = true;
        } else if (Greenfoot.getKey() != null) {
            showWrongImage();
        }
    }
}
```

```
@Override
public void stopped() {
level5Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level5Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level5Song.stop();
}
```

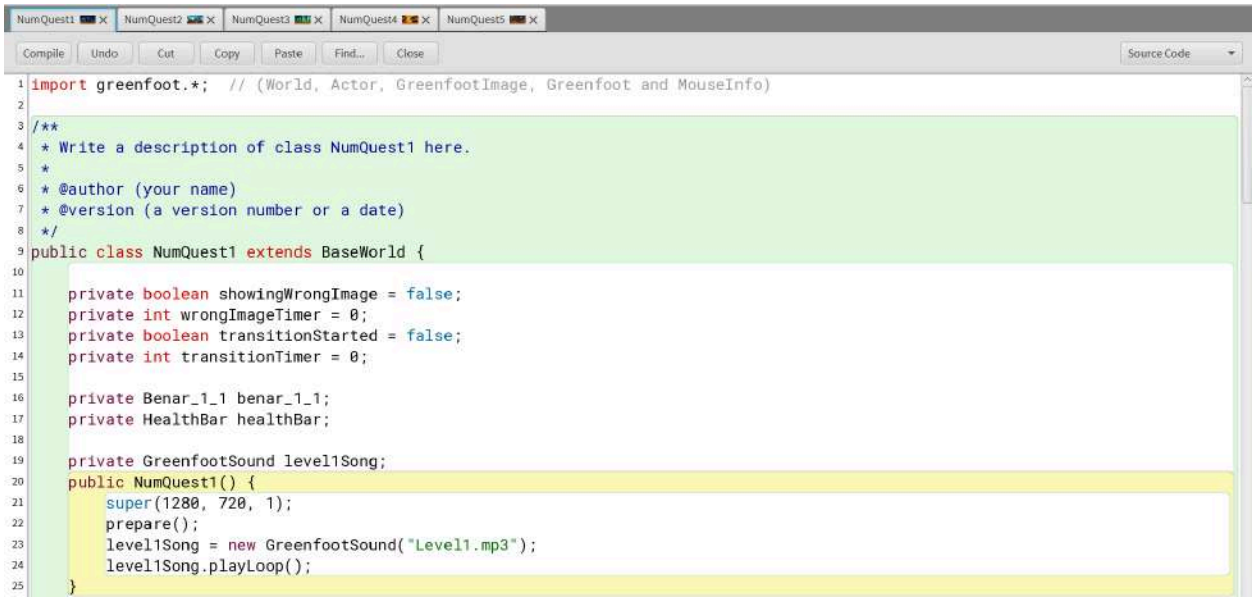
```
    private void prepare() {
        Wayang1 wayang1 = new Wayang1(500, 500);
        addObject(wayang1, 938, 372);
        wayang1.setLocation(1026, 368);

        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
        addObject(gatot_FiLLiT, 178, 399);
        gatot_FiLLiT.setLocation(200, 520);

        Kartini1 = new Kartini1();
        addObject(Kartini1, 421, 373);

        // Inisialisasi health bar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        FiLLiT_5 fiLLiT_5 = new FiLLiT_5(300, 300);
        addObject(fiLLiT_5,722,178);

    }
```

```
    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;

        // Kurangi nyawa
        if (healthBar != null) {
            healthBar.decreaseLife(); // Kurangi health -1
        }
    }
}
```

Setelah menyelesaikan permainan pada world “FiLLiT” maka player akan berpindah ke world puzzle “NumQuest”.

**World dimana player menyelesaikan operasi matematika pertambahan dan pengurangan terdiri dari NumQuest1, NumQuest2, NumQuest3, NumQuest4, NumQuest5.**

**NumQuest1**




```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class NumQuest1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NumQuest1 extends BaseWorld {

    private boolean showingWrongImage = false;
    private int wrongImageTimer = 0;
    private boolean transitionStarted = false;
    private int transitionTimer = 0;

    private Benar_1_1 benar_1_1;
    private HealthBar healthBar;

    private GreenfootSound level1Song;
    public NumQuest1() {
        super(1280, 720, 1);
        prepare();
        level1Song = new GreenfootSound("Level1.mp3");
        level1Song.playLoop();
    }
```

Penjelasan :

```
    private boolean showingWrongImage = false;
    private int wrongImageTimer = 0;
    private boolean transitionStarted = false;
    private int transitionTimer = 0;

    private Benar_1_1 benar_1_1;
    private HealthBar healthBar;

    private GreenfootSound level1Song;
```

> showingWrongImage: Menyimpan status apakah gambar Wrong sedang ditampilkan.
> wrongImageTimer: Menghitung durasi gambar Wrong ditampilkan.
> transitionStarted: Menyimpan status apakah transisi ke world berikutnya telah dimulai.
> transitionTimer: Menghitung durasi transisi.
> benar_1_1: Menyimpan referensi ke objek jawaban benar yang pertama, jika player klik objek ini maka akan menampilkan jawaban benar ke dua yang berwarna hijau menandakan bahwa player benar.
> healthBar: Menyimpan referensi ke HealthBar untuk mengurangi nyawa pemain jika jawaban salah.
> level1Song: Musik yang dimainkan di background world ini.

```
public NumQuest1() {
```

```
        super(1280, 720, 1);
        prepare();
        level1Song = new GreenfootSound("Level1.mp3");
        level1Song.playLoop();
    }
```

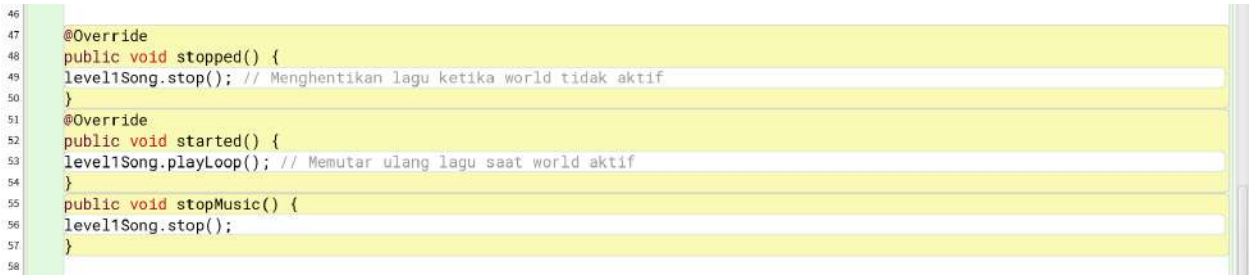
> super(1280, 720, 1): Membuat world dengan ukuran viewport 1280x720 dan skala piksel 1.
> prepare(): Memanggil method untuk menambahkan objek ke world.
> level1Song: Memutar musik level secara berulang (loop).

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 300) { // 5 detik (60 fps * 5)
            removeObjects(getObjects(Wrong.class));
            showingWrongImage = false;
            wrongImageTimer = 0;
        }
    } else if (transitionStarted) {

        transitionTimer++;
        if (transitionTimer >= 300) { // 5 detik (60 fps * 5)
            stopMusic();
            Greenfoot.setWorld(new Labyrntine1());
        }
    } else {
        checkClicks();
    }
}
```

Penjelasan :

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 300) { // 5 detik (60 fps * 5)
            removeObjects(getObjects(Wrong.class));
            showingWrongImage = false;
            wrongImageTimer = 0;
        }
    } else if (transitionStarted) {

        transitionTimer++;
        if (transitionTimer >= 300) { // 5 detik (60 fps * 5)
            stopMusic();
            Greenfoot.setWorld(new Labyrntine1());
        }
    } else {
        checkClicks();
    }
}
```

> if (showingWrongImage): Mengelola timer untuk menampilkan gambar Wrong. Setelah 300 ticks (5 detik), gambar Wrong dihapus.
> else if (transitionStarted): Mengelola transisi ke world berikutnya (dalam hal ini, Labyrntine1) serta menghentikan musik dari world ini setelah 300 ticks (5 detik).
> else: Mengecek interaksi pemain menggunakan method checkClicks().



```
@Override
public void stopped() {
level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level1Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level1Song.stop();
}
```

Penjelasan :

```
@Override
public void stopped() {
level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level1Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level1Song.stop();
}
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

```
59      private void prepare() {
60          Wayang2 wayang2 = new Wayang2(500, 500);
61          addObject(wayang2, 1026, 368);
62
63          int worldWidth = getWidth();
64          int tileWidth = 128;
65          int yPosition = 651;
66
67          for (int x = 0; x < worldWidth; x += tileWidth) {
68              Tiles tile = new Tiles();
69              addObject(tile, x + tileWidth / 2, yPosition);
70          }
71
72          Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
73          addObject(gatot_FiLLiT, 178, 399);
74          gatot_FiLLiT.setLocation(200, 520);
75
76          Soal1 soal1 = new Soal1();
77          addObject(soal1, 530, 176);
78
79          benar_1_1 = new Benar_1_1();
80          addObject(benar_1_1, 626, 312);
81
82          Salah_1_1 salah_1_1 = new Salah_1_1();
83          addObject(salah_1_1, 441, 316);
84
85          Salah_1_2 salah_1_2 = new Salah_1_2();
86          addObject(salah_1_2, 441, 416);
87
88          Salah_1_3 salah_1_3 = new Salah_1_3();
89          addObject(salah_1_3, 626, 412);
90
91          // Inisialisasi HealthBar
92          healthBar = getHealthBar();
93
94          //timer
95          Timer timer = new Timer();
96          addObject(timer, 1180, 50);
97      }
98
```

Penjelasan :

private void prepareLevel() {

…..;

}

> Menambahkan objek yang diperlukan pada world ini sesuai koordinat yang ditentukan.

NOTE : Looping dilakukan untuk menambahkan tile setiap 128 px hingga lebar world tercapai. Tile ini menjadi dasar (ground) di posisi y = 651.

```
    private void checkClicks() {
        if (Greenfoot.mouseClicked(benar_1_1)) {
            benar_1_1.setImage(new GreenfootImage("Benar_1_2.png"));
            transitionStarted = true;
            transitionTimer = 0;
        } else if (Greenfoot.mouseClicked(getObjects(Salah_1_1.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_1_2.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_1_3.class).get(0))) {
            showWrongImage();

            // Kurangi health
            if (healthBar != null) {
                healthBar.decreaseLife();
            }
        }
    }

    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;
    }
}
```

Penjelasan :

```
private void checkClicks() {
      if (Greenfoot.mouseClicked(benar_1_1)) {
         benar_1_1.setImage(new GreenfootImage("Benar_1_2.png"));
         transitionStarted = true;
         transitionTimer = 0;
      } else if (Greenfoot.mouseClicked(getObjects(Salah_1_1.class).get(0)) ||
               Greenfoot.mouseClicked(getObjects(Salah_1_2.class).get(0)) ||
               Greenfoot.mouseClicked(getObjects(Salah_1_3.class).get(0))) {
         showWrongImage();

         // Kurangi health
         if (healthBar != null) {
            healthBar.decreaseLife();
         }
      }
   }
```

> Jika jawaban benar diklik: Gambar jawaban benar_1_1 diubah ke "Benar_1_2.png". Lalu, memulai transisi ke world berikutnya.

> Jika 3 jawaban lain yang salah diklik: Menampilkan gambar salah menggunakan showWrongImage(). Mengurangi nyawa pemain melalui healthBar.decreaseLife().

```
private void showWrongImage() {
      Wrong wrong = new Wrong();
      addObject(wrong, getWidth() / 2, getHeight() / 2);
      showingWrongImage = true;
      wrongImageTimer = 0;
   }
```

> Menambahkan objek Wrong di tengah layar.
> Mengatur status showingWrongImage menjadi true untuk memulai timer penghapusan gambar.

**NumQuest2, NumQuest3, NumQuest4, NumQuest5 memiliki kode Java yang sama seperti NumQuest1, perbedaannya hanya terdapat pada :**

1. Musik
    - NumQuest2 : level2Song = new GreenfootSound("Level2.mp3");
    - NumQuest3 : level3Song = new GreenfootSound("Level3.mp3");
    - NumQuest4 : level4Song = new GreenfootSound("Level4.mp3");
    - NumQuest5 : level5Song = new GreenfootSound("th06_06.mp3");
2. Penempatan objek
   > Soal pada setiap world NumQuest berbeda-beda, oleh karena itu objek gambar yang ditempatkan juga berbeda pada setiap world.
3. Background (Background diset langsung melalui setimage pada setiap world NumQuest)
4. Setiap NumQuest memiliki 1 objek jawaban benar dan 3 objek jawaban salah yang berbeda-beda

**NumQuest2**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class NumQuest1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NumQuest2 extends BaseWorld {

    private boolean showingWrongImage = false;
    private int wrongImageTimer = 0;
    private boolean transitionStarted = false;
    private int transitionTimer = 0;

    private Benar_2_1 benar_2_1;
    private HealthBar healthBar; // Health bar

    private GreenfootSound level2Song;
    public NumQuest2() {
        super(1280, 720, 1);
        prepare();

        level2Song = new GreenfootSound("Level2.mp3");
        level2Song.playLoop();
    }
```

```java
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 300) { // 5 detik (60 fps * 5)
            removeObjects(getObjects(Wrong.class));
            showingWrongImage = false;
            wrongImageTimer = 0;
        }
    } else if (transitionStarted) {
        transitionTimer++;
        if (transitionTimer >= 300) { // 5 detik (60 fps * 5)
            stopMusic();
            Greenfoot.setWorld(new Labyrntine2());
        }
    } else {
        checkClicks();
    }
}

@Override
public void stopped() {
level2Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level2Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level2Song.stop();
}
```

```java
private void prepare() {
    Wayang2 wayang2 = new Wayang2(500, 500);
    addObject(wayang2, 1026, 368);

    int worldWidth = getWidth();
    int tileWidth = 128;
    int yPosition = 651;

    for (int x = 0; x < worldWidth; x += tileWidth) {
        Tiles tile = new Tiles();
        addObject(tile, x + tileWidth / 2, yPosition);
    }

    Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
    addObject(gatot_FiLLiT, 178, 399);
    gatot_FiLLiT.setLocation(200, 520);

    Soal2 soal2 = new Soal2();
    addObject(soal2, 530, 176);

    benar_2_1 = new Benar_2_1();
    addObject(benar_2_1, 441, 316);

    Salah_2_1 salah_2_1 = new Salah_2_1();
    addObject(salah_2_1, 626, 312);

    Salah_2_2 salah_2_2 = new Salah_2_2();
    addObject(salah_2_2, 441, 416);

    Salah_2_3 salah_2_3 = new Salah_2_3();
    addObject(salah_2_3, 626, 412);
```

```
        // Inisialisasi HealthBar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world
    }

    private void checkClicks() {
        if (Greenfoot.mouseClicked(benar_2_1)) {
            benar_2_1.setImage(new GreenfootImage("Benar_2_2.png"));
            transitionStarted = true;
            transitionTimer = 0;
        } else if (Greenfoot.mouseClicked(getObjects(Salah_2_1.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_2_2.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_2_3.class).get(0))) {
            showWrongImage();

            // Kurangi health
            if (healthBar != null) {
                healthBar.decreaseLife();
            }
        }
    }

    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;
    }
}
```

## NumQuest3

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class NumQuest1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NumQuest3 extends BaseWorld {

    private boolean showingWrongImage = false;
    private int wrongImageTimer = 0;
    private boolean transitionStarted = false;
    private int transitionTimer = 0;

    private Benar_3_1 benar_3_1;
    private HealthBar healthBar; // Health bar

    private GreenfootSound level3Song;
    public NumQuest3() {
        super(1280, 720, 1);
        prepare();
        level3Song = new GreenfootSound("Level3.mp3");
        level3Song.playLoop();
    }
```

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 300) { // 5 detik (60 fps * 5)
            removeObjects(getObjects(Wrong.class));
            showingWrongImage = false;
            wrongImageTimer = 0;
        }
    } else if (transitionStarted) {
        transitionTimer++;
        if (transitionTimer >= 300) { // 5 detik (60 fps * 5)
            stopMusic();
            Greenfoot.setWorld(new Labyrntine3());
        }
    } else {
        checkClicks();
    }
}

@Override
public void stopped() {
level3Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level3Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level3Song.stop();
}
```

```
private void prepare() {
    Wayang2 wayang2 = new Wayang2(500, 500);
    addObject(wayang2, 1026, 368);

    int worldWidth = getWidth();
    int tileWidth = 128;
    int yPosition = 651;

    for (int x = 0; x < worldWidth; x += tileWidth) {
        Tiles tile = new Tiles();
        addObject(tile, x + tileWidth / 2, yPosition);
    }

    Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
    addObject(gatot_FiLLiT, 178, 399);
    gatot_FiLLiT.setLocation(200, 520);

    Soal3 soal3 = new Soal3();
    addObject(soal3, 530, 176);

    benar_3_1 = new Benar_3_1();
    addObject(benar_3_1, 626, 412);

    Salah_3_1 salah_3_1 = new Salah_3_1();
    addObject(salah_3_1, 441, 316);

    Salah_3_2 salah_3_2 = new Salah_3_2();
    addObject(salah_3_2, 626, 312);

    Salah_3_3 salah_3_3 = new Salah_3_3();
    addObject(salah_3_3, 441, 416);
```

```
        // Inisialisasi HealthBar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world
    }

    private void checkClicks() {
        if (Greenfoot.mouseClicked(benar_3_1)) {
            benar_3_1.setImage(new GreenfootImage("Benar_3_2.png"));
            transitionStarted = true;
            transitionTimer = 0;
        } else if (Greenfoot.mouseClicked(getObjects(Salah_3_1.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_3_2.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_3_3.class).get(0))) {
            showWrongImage();

            // Kurangi health
            if (healthBar != null) {
                healthBar.decreaseLife();
            }
        }
    }

    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;
    }
}
```

## NumQuest4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class NumQuest4 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NumQuest4 extends BaseWorld {

    private boolean showingWrongImage = false;
    private int wrongImageTimer = 0;
    private boolean transitionStarted = false;
    private int transitionTimer = 0;

    private Benar_4_1 benar_4_1;
    private HealthBar healthBar; // Health bar

    private GreenfootSound level4Song;
    public NumQuest4() {
        super(1280, 720, 1);
        prepare();
        level4Song = new GreenfootSound("Level4.mp3");
        level4Song.playLoop();
    }
```

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 300) { // 5 detik (60 fps * 5)
            removeObjects(getObjects(Wrong.class));
            showingWrongImage = false;
            wrongImageTimer = 0;
        }
    } else if (transitionStarted) {
        transitionTimer++;
        if (transitionTimer >= 300) { // 5 detik (60 fps * 5)
            stopMusic();
            Greenfoot.setWorld(new Labyrntine4());
        }
    } else {
        checkClicks();
    }
}

@Override
public void stopped() {
level4Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level4Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level4Song.stop();
}
```

```
private void prepare() {
    Wayang2 wayang2 = new Wayang2(500, 500);
    addObject(wayang2, 1026, 368);

    int worldWidth = getWidth();
    int tileWidth = 128;
    int yPosition = 651;

    for (int x = 0; x < worldWidth; x += tileWidth) {
        Tiles tile = new Tiles();
        addObject(tile, x + tileWidth / 2, yPosition);
    }

    Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
    addObject(gatot_FiLLiT, 178, 399);
    gatot_FiLLiT.setLocation(200, 520);

    Soal4 soal4 = new Soal4();
    addObject(soal4, 530, 176);

    benar_4_1 = new Benar_4_1();
    addObject(benar_4_1, 626, 312);

    Salah_4_1 salah_4_1 = new Salah_4_1();
    addObject(salah_4_1, 441, 316);

    Salah_4_2 salah_4_2 = new Salah_4_2();
    addObject(salah_4_2, 441, 416);

    Salah_4_3 salah_4_3 = new Salah_4_3();
    addObject(salah_4_3, 626, 412);
```

```
        // Inisialisasi HealthBar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world
    }

    private void checkClicks() {
        if (Greenfoot.mouseClicked(benar_4_1)) {
            benar_4_1.setImage(new GreenfootImage("Benar_4_2.png"));
            transitionStarted = true;
            transitionTimer = 0;
        } else if (Greenfoot.mouseClicked(getObjects(Salah_4_1.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_4_2.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_4_3.class).get(0))) {
            showWrongImage();

            // Kurangi health
            if (healthBar != null) {
                healthBar.decreaseLife();
            }
        }
    }

    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;
    }
}
```

## NumQuest5

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class NumQuest5 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NumQuest5 extends BaseWorld {

    private boolean showingWrongImage = false;
    private int wrongImageTimer = 0;
    private boolean transitionStarted = false;
    private int transitionTimer = 0;

    private Benar_5_1 benar_5_1;
    private HealthBar healthBar; // Health bar

    private GreenfootSound level5Song;
    public NumQuest5() {
        super(1280, 720, 1);
        prepare();
        level5Song = new GreenfootSound("th06_06.mp3");
        level5Song.playLoop();
    }
```

```
public void act() {
    if (showingWrongImage) {
        wrongImageTimer++;
        if (wrongImageTimer >= 300) { // 5 detik (60 fps * 5)
            removeObjects(getObjects(Wrong.class));
            showingWrongImage = false;
            wrongImageTimer = 0;
        }
    } else if (transitionStarted) {
        transitionTimer++;
        if (transitionTimer >= 300) { // 5 detik (60 fps * 5)
            stopMusic();
            Greenfoot.setWorld(new Labyrntine5());
        }
    } else {
        checkClicks();
    }
}

@Override
public void stopped() {
level15Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level15Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level15Song.stop();
}
```

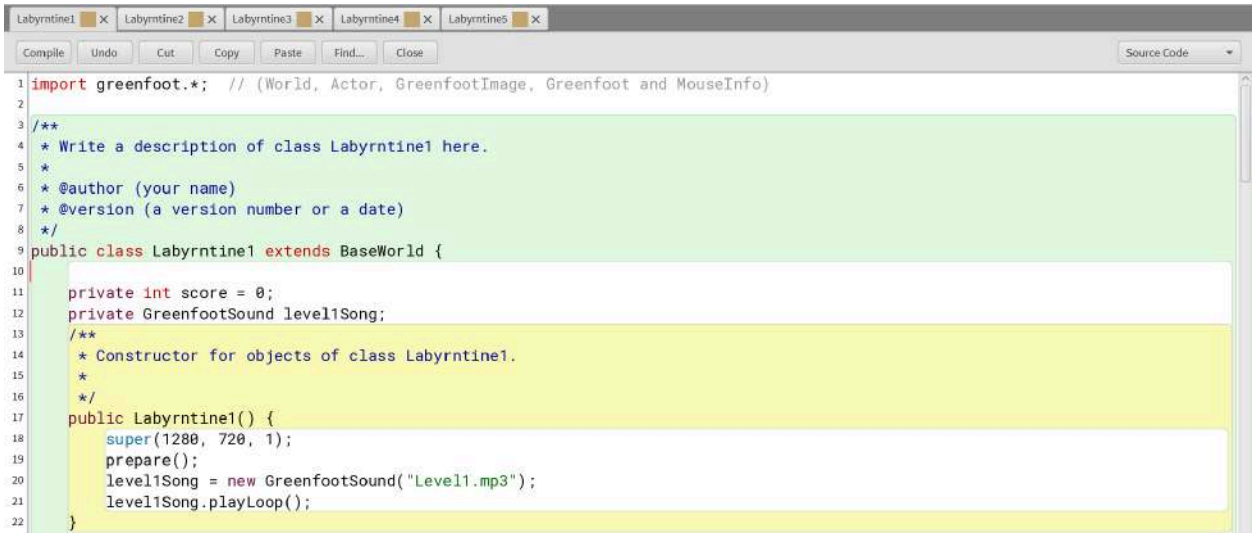
```
private void prepare() {
    Wayang2 wayang2 = new Wayang2(500, 500);
    addObject(wayang2, 1026, 368);

    int worldWidth = getWidth();
    int tileWidth = 128;
    int yPosition = 651;

    for (int x = 0; x < worldWidth; x += tileWidth) {
        Tiles tile = new Tiles();
        addObject(tile, x + tileWidth / 2, yPosition);
    }

    Gatot_FiLLiT gatot_FiLLiT = new Gatot_FiLLiT(90, 130);
    addObject(gatot_FiLLiT, 178, 399);
    gatot_FiLLiT.setLocation(200, 520);

    Soal5 soal5 = new Soal5();
    addObject(soal5, 530, 176);

    benar_5_1 = new Benar_5_1();
    addObject(benar_5_1, 441, 416);

    Salah_5_1 salah_5_1 = new Salah_5_1();
    addObject(salah_5_1, 441, 316);

    Salah_5_2 salah_5_2 = new Salah_5_2();
    addObject(salah_5_2, 626, 312);

    Salah_5_3 salah_5_3 = new Salah_5_3();
    addObject(salah_5_3, 626, 412);
```

```
        // Inisialisasi HealthBar
        healthBar = getHealthBar();

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world
    }

    private void checkClicks() {
        if (Greenfoot.mouseClicked(benar_5_1)) {
            benar_5_1.setImage(new GreenfootImage("Benar_5_2.png"));
            transitionStarted = true;
            transitionTimer = 0;
        } else if (Greenfoot.mouseClicked(getObjects(Salah_5_1.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_5_2.class).get(0)) ||
                   Greenfoot.mouseClicked(getObjects(Salah_5_3.class).get(0))) {
            showWrongImage();

            // Kurangi health
            if (healthBar != null) {
                healthBar.decreaseLife();
            }
        }
    }

    private void showWrongImage() {
        Wrong wrong = new Wrong();
        addObject(wrong, getWidth() / 2, getHeight() / 2);
        showingWrongImage = true;
        wrongImageTimer = 0;
    }
}
```

Setelah menyelesaikan permainan pada world “NumQuest” maka player akan berpindah ke world puzzle “Labyrntine”.

**World dimana player mengasah kemampuan berpikir HOTSnya dengan puzzle labirin, terdiri dari Labyrntine1, Labyrntine2, Labyrntine3, Labyrntine4, Labyrntine5.**

**Labyrntine1**




```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Labyrntine1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Labyrntine1 extends BaseWorld {

    private int score = 0;
    private GreenfootSound level1Song;
    /**
     * Constructor for objects of class Labyrntine1.
     *
     */
    public Labyrntine1() {
        super(1280, 720, 1);
        prepare();
        level1Song = new GreenfootSound("Level1.mp3");
        level1Song.playLoop();
    }
```

Penjelasan :

```
private int score = 0;
private GreenfootSound level1Song;
```

> score: Variabel untuk mencatat skor pemain, dari 0
> level1Song: Objek GreenfootSound untuk memutar musik latar secara berulang.

```
public Labyrntine1() {
        super(1280, 720, 1);
        prepare();
        level1Song = new GreenfootSound("Level1.mp3");
        level1Song.playLoop();
    }
```

> super(1280, 720, 1): Membuat world dengan ukuran viewport 1280x720 dan skala piksel 1.
> prepare(): Memanggil method untuk menambahkan objek ke world.
> level1Song: Memanggil musik level secara berulang (loop).

```
23
24      @Override
25      public void stopped() {
26      level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
27      }
28      @Override
29      public void started() {
30      level1Song.playLoop(); // Memutar ulang lagu saat world aktif
31      }
32      public void stopMusic() {
33      level1Song.stop();
34      }
35
```

Penjelasan :

```
 @Override
    public void stopped() {
    level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level1Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level1Song.stop();
    }
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

```
    /**
     * Prepare the world for the start of the program.
     * That is: create the initial objects and add them to the world.
     */
    private void prepare() {
        // Crate1
        int crateWidth = 77;
        int crateHeight = 77;

        // horizontal crates
        for (int x = 232; x <= 1216; x += crateWidth) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight); // Scale each crate
            addObject(crate, x, 146);
        }

        // vertical crates (left)
        for (int y = 146; y <= 661; y += crateHeight) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, 78, y);
        }

        // vertical crates (right)
        for (int y = 223; y <= 608; y += crateHeight) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, 1156, y);
        }

        // horizontal crates (bottom)
        for (int x = 78; x <= 1079; x += crateWidth) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, x, 608);
        }

        // Crates tantangan
        Crate crate1 = new Crate();
        scaleActor(crate1, crateWidth, crateHeight);
        addObject(crate1, 540, 320);

        Crate crate2 = new Crate();
        scaleActor(crate2, crateWidth, crateHeight);
        addObject(crate2, 620, 320);

        Crate crate3 = new Crate();
        scaleActor(crate3, crateWidth, crateHeight);
        addObject(crate3, 700, 320);

        Crate crate4 = new Crate();
        scaleActor(crate4, crateWidth, crateHeight);
        addObject(crate4, 700, 430);

        Crate crate5 = new Crate();
        scaleActor(crate5, crateWidth, crateHeight);
        addObject(crate5, 620, 430);

        // player
        Gatot_Labyrntine gatot_Labyrntine = new Gatot_Labyrntine(60, 90);
        scaleActor(gatot_Labyrntine, 60, 90);
        addObject(gatot_Labyrntine, 390, 520);

        //finish
        Finish_Labyrntine finish_Labyrntine = new Finish_Labyrntine();
        scaleActor(finish_Labyrntine, 50, 50);
        addObject(finish_Labyrntine, 158, 143);

        // coin
        Coin coin1 = new Coin();
        scaleActor(coin1, 100, 100);
        addObject(coin1, 450, 230);

        Coin coin2 = new Coin();
        scaleActor(coin2, 100, 100);
        addObject(coin2, 900, 350);
```

```
        Coin coin3 = new Coin();
        scaleActor(coin3, 100, 100);
        addObject(coin3, 170, 400);

        Coin coin4 = new Coin();
        scaleActor(coin4, 100, 100);
        addObject(coin4, 500, 500);

        Coin coin5 = new Coin();
        scaleActor(coin5, 100, 100);
        addObject(coin5, 700, 530);

        // timer
        Timer timer = new Timer();
        addObject(timer, 1180, 50);

        Crate crate46 = new Crate();
        addObject(crate46,394,389);
        coin2.setLocation(848,234);
        Crate crate47 = new Crate();
        addObject(crate47,858,529);
    }
```

Penjelasan :

```
private void prepareLevel() {
        …..;
}
```

> Menambahkan objek yang diperlukan pada world ini sesuai koordinat yang ditentukan.

NOTE :

> crateWidth dan crateHeight dideklarasikan menjadi 77x77 piksel.

> horizontal crates: Loop for berjalan dari x = 232 hingga x = 1216 dengan langkah sebesar crateWidth (77). Menempatkan Crate secara horizontal di koordinat y = 146 dengan jarak tetap sebesar 77 piksel di antara Crate.

> vertical crates (left): Loop for berjalan dari y = 146 hingga y = 661 dengan langkah sebesar crateHeight (77). Crates ditempatkan pada x = 78.

> vertical crates (right): Loop for berjalan dari y = 223 hingga y = 608 dengan langkah sebesar crateHeight (77). Crates ditempatkan pada x = 1156.

> horizontal crates (bottom): Loop for berjalan dari x = 78 hingga x = 1079 dengan langkah sebesar crateWidth (77). Crates ditempatkan pada y = 608.

```

    private void scaleActor(Actor actor, int width, int height) {
        GreenfootImage image = actor.getImage();
        image.scale(width, height);
        actor.setImage(image);
    }

    public void addScore(int points) {
        score += points;
        System.out.println("Score: " + score);
    }
}
```

Penjelasan :

```
private void scaleActor(Actor actor, int width, int height) {
      GreenfootImage image = actor.getImage();
      image.scale(width, height);
      actor.setImage(image);
   }
```

> Method agar objek actor dapat diatur manual ukurannya sesuai dengan dimensi yang diinginkan.

```
public void addScore(int points) {
      score += points;
      System.out.println("Score: " + score);
   }
```

> Menambah skor ketika pemain mengumpulkan koin.

**Labyrntine2, Labyrntine3, Labyrntine4, Labyrntine5 memiliki kode Java yang sama seperti Labyrntine1, perbedaannya hanya terdapat pada :**

1. Musik
    - Labyrntine2 : level2Song = new GreenfootSound("Level2.mp3");
    - Labyrntine3 : level3Song = new GreenfootSound("Level3.mp3");
    - Labyrntine4 : level4Song = new GreenfootSound("Level4.mp3");
    - Labyrntine5 : level5Song = new GreenfootSound("th06_06.mp3");
2. Penempatan objek

    > Crate, Coin, dan Finish_Labyrntine yang diletakkan pada setiap world memiliki koordinat yang berbeda-beda sesuai kebutuhan world agar permainan semakin menarik dan melatih HOTS pemain.

**Labyrinthine2**

```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Labyrntine1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Labyrntine2 extends BaseWorld {

    private int score = 0; // Initialize the score
    private GreenfootSound level2Song;
    /**
     * Constructor for objects of class Labyrntine1.
     *
     */
    public Labyrntine2() {
        super(1280, 720, 1);
        prepare();

        level2Song = new GreenfootSound("Level2.mp3");
        level2Song.playLoop();
    }

    @Override
    public void stopped() {
    level2Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level2Song.playLoop(); // Memutar ulang lagu saat world aktif
    }

    public void stopMusic() {
    level2Song.stop();
    }

    /**
     * Prepare the world for the start of the program.
     * That is: create the initial objects and add them to the world.
     */
    private void prepare() {
        // Existing crates layout
        int crateWidth = 77;
        int crateHeight = 77;

        // Add horizontal crates
        for (int x = 155; x <= 1216; x += crateWidth) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight); // Scale each crate
            addObject(crate, x, 146);
        }

        // Add vertical crates (left)
        for (int y = 146; y <= 661; y += crateHeight) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, 78, y);
        }
```

```
    // Add vertical crates (right)
    for (int y = 223; y <= 377; y += crateHeight) {
        Crate crate = new Crate();
        scaleActor(crate, crateWidth, crateHeight);
        addObject(crate, 1156, y);
    }

    // Add horizontal crates (bottom)
    for (int x = 78; x <= 1156; x += crateWidth) {
        Crate crate = new Crate();
        scaleActor(crate, crateWidth, crateHeight);
        addObject(crate, x, 608);
    }
    // Add player character
    Gatot_Labyrntine gatot_Labyrntine = new Gatot_Labyrntine(60, 90);
    scaleActor(gatot_Labyrntine, 60, 90); // Scale the player
    addObject(gatot_Labyrntine, 159, 237);

    // Add finish point
    Finish_Labyrntine2 finish_Labyrntine2 = new Finish_Labyrntine2();
    scaleActor(finish_Labyrntine2, 50, 50); // Scale finish point
    addObject(finish_Labyrntine2, 1156, 531);

    // Add coins for scoring
    Coin coin1 = new Coin();
    scaleActor(coin1, 100, 100); // Scale coin to 100x100
    addObject(coin1, 450, 230);

    Coin coin2 = new Coin();
    scaleActor(coin2, 100, 100); // Scale coin to 100x100
    addObject(coin2, 930, 347);
```

```
    Coin coin3 = new Coin();
    scaleActor(coin3, 100, 100); // Scale coin to 100x100
    addObject(coin3, 620, 515);

    Coin coin4 = new Coin();
    scaleActor(coin4, 100, 100); // Scale coin to 100x100
    addObject(coin4, 782, 329);

    // Add timer
    Timer timer = new Timer();
    addObject(timer, 1180, 50);

    Crate crate41 = new Crate();
    addObject(crate41,154,532);
    Crate crate42 = new Crate();
    addObject(crate42,310,220);
    Crate crate43 = new Crate();
    addObject(crate43,537,430);
    Crate crate44 = new Crate();
    addObject(crate44,536,507);
    Crate crate45 = new Crate();
    addObject(crate45,364,368);
    Crate crate46 = new Crate();
    addObject(crate46,848,530);
    Crate crate47 = new Crate();
    addObject(crate47,697,329);
    Crate crate48 = new Crate();
    addObject(crate48,1156,435);
}
```

```
    /**
     * Scale an actor's image to the given width and height.
     */
    private void scaleActor(Actor actor, int width, int height) {
        GreenfootImage image = actor.getImage();
        image.scale(width, height);
        actor.setImage(image);
    }

    /**
     * Update the score when a coin is collected.
     */
    public void addScore(int points) {
        score += points;
        System.out.println("Score: " + score);
    }
}
```

**Labyrinthine3**

```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Labyrntine1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Labyrntine3 extends BaseWorld {

    private int score = 0; // Initialize the score
    private GreenfootSound level3Song;
    /**
     * Constructor for objects of class Labyrntine1.
     *
     */
    public Labyrntine3() {
        super(1280, 720, 1);
        prepare();
        level3Song = new GreenfootSound("Level3.mp3");
        level3Song.playLoop();
    }

    @Override
    public void stopped() {
    level3Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level3Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
```

```java
    public void stopMusic() {
    level3Song.stop();
    }

    /**
     * Prepare the world for the start of the program.
     * That is: create the initial objects and add them to the world.
     */
    private void prepare() {
        // Existing crates layout
        int crateWidth = 77;
        int crateHeight = 77;

        // Add horizontal crates
        for (int x = 155; x <= 1216; x += crateWidth) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight); // Scale each crate
            addObject(crate, x, 146);
        }

        // Add vertical crates (left)
        for (int y = 146; y <= 661; y += crateHeight) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, 78, y);
        }
```

```
        // Add vertical crates (right)
        for (int y = 223; y <= 608; y += crateHeight) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, 1156, y);
        }

        // Add horizontal crates (bottom)
        for (int x = 232; x <= 1156; x += crateWidth) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, x, 608);
        }
        // Add player character
        Gatot_Labyrntine gatot_Labyrntine = new Gatot_Labyrntine(60, 90);
        scaleActor(gatot_Labyrntine, 60, 90); // Scale the player
        addObject(gatot_Labyrntine, 846, 234);

        // Add finish point
        Finish_Labyrntine3 finish_Labyrntine3 = new Finish_Labyrntine3();
        scaleActor(finish_Labyrntine3, 50, 50); // Scale finish point
        addObject(finish_Labyrntine3, 155, 608);

        // Add coins for scoring
        Coin coin1 = new Coin();
        scaleActor(coin1, 100, 100); // Scale coin to 100x100
        addObject(coin1, 450, 230);

        Coin coin2 = new Coin();
        scaleActor(coin2, 100, 100); // Scale coin to 100x100
        addObject(coin2, 156, 527);

        Coin coin3 = new Coin();
        scaleActor(coin3, 100, 100); // Scale coin to 100x100
        addObject(coin3, 694,410);

        Coin coin4 = new Coin();
        scaleActor(coin4, 100, 100); // Scale coin to 100x100
        addObject(coin4, 933, 333);

        // Add timer
        Timer timer = new Timer();
        addObject(timer, 1180, 50);

        Crate crate41 = new Crate();
        addObject(crate41,152,224);
        Crate crate42 = new Crate();
        addObject(crate42,1076,377);
        Crate crate43 = new Crate();
        addObject(crate43,612,435);
        Crate crate44 = new Crate();
        addObject(crate44,330,355);
        Crate crate45 = new Crate();
        addObject(crate45,409,355);
        Crate crate46 = new Crate();
        addObject(crate46,329,434);
        Crate crate47 = new Crate();
        addObject(crate47,408,433);
        Crate crate48 = new Crate();
        addObject(crate48,484,389);
    }
```

```
    /**
     * Scale an actor's image to the given width and height.
     */
    private void scaleActor(Actor actor, int width, int height) {
        GreenfootImage image = actor.getImage();
        image.scale(width, height);
        actor.setImage(image);
    }

    /**
     * Update the score when a coin is collected.
     */
    public void addScore(int points) {
        score += points;
        System.out.println("Score: " + score);
    }
}
```

## Labyrinthine4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Labyrntine1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Labyrntine4 extends BaseWorld {

    private int score = 0; // Initialize the score
    private GreenfootSound level4Song;
    /**
     * Constructor for objects of class Labyrntine1.
     *
     */
    public Labyrntine4() {
        super(1280, 720, 1);
        prepare();
        level4Song = new GreenfootSound("Level4.mp3");
        level4Song.playLoop();
    }

    @Override
    public void stopped() {
    level4Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level4Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
```

```
public void stopMusic() {
level4Song.stop();
}

/**
 * Prepare the world for the start of the program.
 * That is: create the initial objects and add them to the world.
 */
private void prepare() {
    // Existing crates layout
    int crateWidth = 77;
    int crateHeight = 77;

    // Add horizontal crates
    for (int x = 155; x <= 1216; x += crateWidth) {
        Crate crate = new Crate();
        scaleActor(crate, crateWidth, crateHeight); // Scale each crate
        addObject(crate, x, 146);
    }

    // Add vertical crates (left)
    for (int y = 146; y <= 661; y += crateHeight) {
        Crate crate = new Crate();
        scaleActor(crate, crateWidth, crateHeight);
        addObject(crate, 78, y);
    }
```

```
    // Add vertical crates (right)
    for (int y = 377; y <= 608; y += crateHeight) {
        Crate crate = new Crate();
        scaleActor(crate, crateWidth, crateHeight);
        addObject(crate, 1156, y);
    }

    // Add horizontal crates (bottom)
    for (int x = 78; x <= 1079; x += crateWidth) {
        Crate crate = new Crate();
        scaleActor(crate, crateWidth, crateHeight);
        addObject(crate, x, 608);
    }

    // Add player character
    Gatot_Labyrntine gatot_Labyrntine = new Gatot_Labyrntine(60, 90);
    scaleActor(gatot_Labyrntine, 60, 90); // Scale the player
    addObject(gatot_Labyrntine, 1080, 520);

    // Add finish point
    Finish_Labyrntine4 finish_Labyrntine4 = new Finish_Labyrntine4();
    scaleActor(finish_Labyrntine4, 50, 50); // Scale finish point
    addObject(finish_Labyrntine4, 1156, 223);

    // Add coins for scoring
    Coin coin1 = new Coin();
    scaleActor(coin1, 100, 100); // Scale coin to 100x100
    addObject(coin1, 450, 230);

    Coin coin5 = new Coin();
    scaleActor(coin5, 100, 100); // Scale coin to 100x100
    addObject(coin5, 627, 371);
```

```
        Coin coin2 = new Coin();
        scaleActor(coin2, 100, 100); // Scale coin to 100x100
        addObject(coin2, 165, 395);

        Coin coin3 = new Coin();
        scaleActor(coin3, 100, 100); // Scale coin to 100x100
        addObject(coin3, 510, 512);

        Coin coin4 = new Coin();
        scaleActor(coin4, 100, 100); // Scale coin to 100x100
        addObject(coin4, 998, 234);

        // Add timer
        Timer timer = new Timer();
        addObject(timer, 1180, 50);
        Crate crate41 = new Crate();
        addObject(crate41,1077,322);
        Crate crate42 = new Crate();
        addObject(crate42,1076,400);
        Crate crate43 = new Crate();
        addObject(crate43,997,322);
        Crate crate44 = new Crate();
        addObject(crate44,538,410);
        Crate crate45 = new Crate();
        addObject(crate45,538,350);
        Crate crate46 = new Crate();
        addObject(crate46,262,397);
        Crate crate47 = new Crate();
        addObject(crate47,338,397);
        Crate crate48 = new Crate();
        addObject(crate48,414,397);
        Crate crate49 = new Crate();
        addObject(crate49,359,221);
        Crate crate50 = new Crate();
        addObject(crate50,1155,323);
    }

    /**
     * Scale an actor's image to the given width and height.
     */
    private void scaleActor(Actor actor, int width, int height) {
        GreenfootImage image = actor.getImage();
        image.scale(width, height);
        actor.setImage(image);
    }

    /**
     * Update the score when a coin is collected.
     */
    public void addScore(int points) {
        score += points;
        System.out.println("Score: " + score);
    }
}
```

## Labyrinthine5

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Labyrntine1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Labyrntine5 extends BaseWorld {

    private int score = 0; // Initialize the score
    private GreenfootSound level5Song;
    /**
     * Constructor for objects of class Labyrntine1.
     *
     */
    public Labyrntine5() {
        super(1280, 720, 1);
        prepare();
        level5Song = new GreenfootSound("th06_06.mp3");
        level5Song.playLoop();
    }

    @Override
    public void stopped() {
    level5Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level5Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
```

```
    public void stopMusic() {
    level5Song.stop();
    }

    /**
     * Prepare the world for the start of the program.
     * That is: create the initial objects and add them to the world.
     */
    private void prepare() {
        // Existing crates layout
        int crateWidth = 77;
        int crateHeight = 77;

        // Add horizontal crates
        for (int x = 155; x <= 1216; x += crateWidth) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight); // Scale each crate
            addObject(crate, x, 146);
        }

        // Add vertical crates (left)
        for (int y = 146; y <= 661; y += crateHeight) {
            Crate crate = new Crate();
            scaleActor(crate, crateWidth, crateHeight);
            addObject(crate, 78, y);
        }
```

```
// Add vertical crates (right)
for (int y = 223; y <= 608; y += crateHeight) {
    Crate crate = new Crate();
    scaleActor(crate, crateWidth, crateHeight);
    addObject(crate, 1156, y);
}

// Add horizontal crates (bottom)
for (int x = 617; x <= 1156; x += crateWidth) {
    Crate crate = new Crate();
    scaleActor(crate, crateWidth, crateHeight);
    addObject(crate, x, 608);
}

// Add additional horizontal crates (bottom)
for (int x = 155; x <= 463; x += crateWidth) {
    Crate crate = new Crate();
    scaleActor(crate, crateWidth, crateHeight);
    addObject(crate, x, 608);
}
// Add player character
Gatot_Labyrntine gatot_Labyrntine = new Gatot_Labyrntine(60, 90);
scaleActor(gatot_Labyrntine, 60, 90); // Scale the player
addObject(gatot_Labyrntine, 686, 234);

// Add finish point
Finish_Labyrntine5 finish_Labyrntine5 = new Finish_Labyrntine5();
scaleActor(finish_Labyrntine5, 50, 50); // Scale finish point
addObject(finish_Labyrntine5, 540, 608);
```

```
// Add coins for scoring
Coin coin1 = new Coin();
scaleActor(coin1, 100, 100); // Scale coin to 100x100
addObject(coin1, 450, 230);

Coin coin5 = new Coin();
scaleActor(coin5, 100, 100); // Scale coin to 100x100
addObject(coin5, 832, 234);

Coin coin2 = new Coin();
scaleActor(coin2, 100, 100); // Scale coin to 100x100
addObject(coin2, 166, 232);

Coin coin3 = new Coin();
scaleActor(coin3, 100, 100); // Scale coin to 100x100
addObject(coin3, 542, 360);

Coin coin4 = new Coin();
scaleActor(coin4, 100, 100); // Scale coin to 100x100
addObject(coin4, 864, 514);

// Add timer
Timer timer = new Timer();
addObject(timer, 1180, 50);
Crate crate41 = new Crate();
addObject(crate41,460,353);
Crate crate42 = new Crate();
addObject(crate42,256,275);
Crate crate43 = new Crate();
addObject(crate43,1074,250);
Crate crate44 = new Crate();
addObject(crate44,994,250);
```

```
        Crate crate45 = new Crate();
        addObject(crate45,913,250);
    }

    /**
     * Scale an actor's image to the given width and height.
     */
    private void scaleActor(Actor actor, int width, int height) {
        GreenfootImage image = actor.getImage();
        image.scale(width, height);
        actor.setImage(image);
    }

    /**
     * Update the score when a coin is collected.
     */
    public void addScore(int points) {
        score += points;
        System.out.println("Score: " + score);
    }
}
```

Setelah menyelesaikan permainan di world “Labyrntine” maka player akan berpindah ke world terakhir pada setiap level, yaitu “BossFight”.

**World dimana player harus menghadapi satu boss per pada setiap level untuk bisa lanjut ke tahap berikutnya, terdiri dari BossFight1, BossFight2, BossFight3, BossFight4, BossFight5.**

**BossFight1**


Penjelasan :

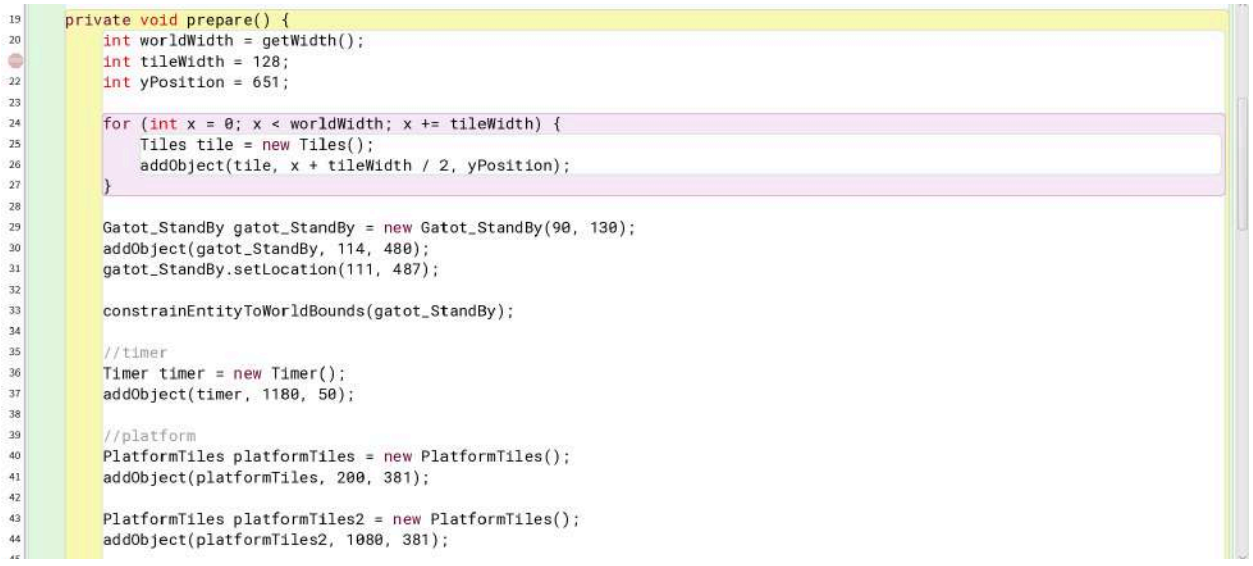
private GreenfootSound level1Song;

> Variabel untuk menyimpan musik pada level ini.

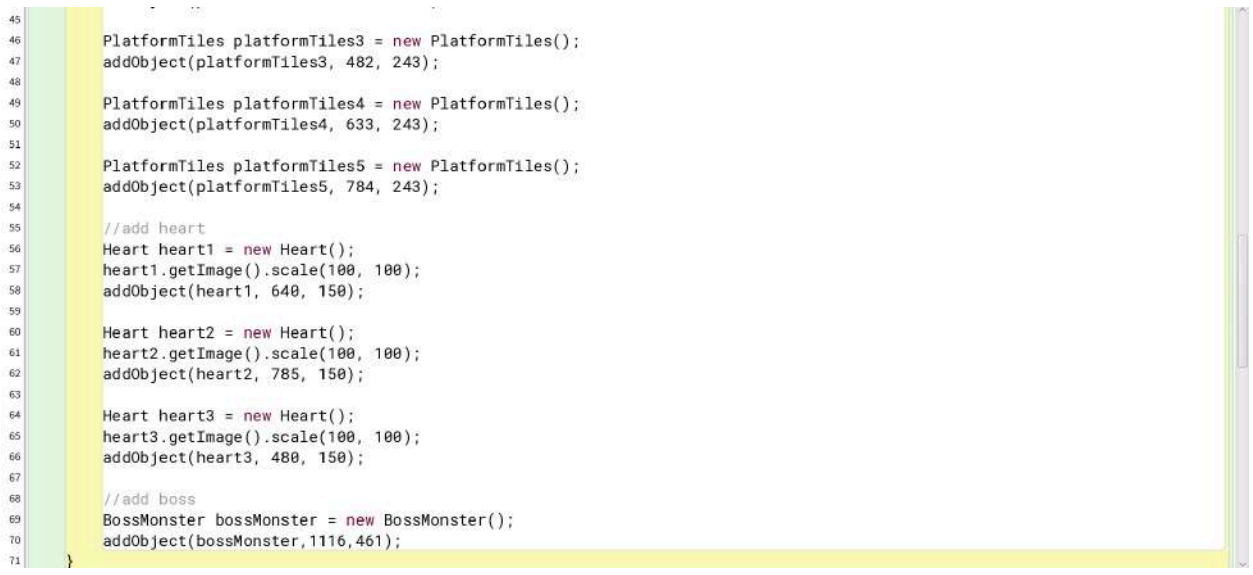
public BossFight1() {
    super(1280, 720, 1);

```
        prepare();

        level1Song = new GreenfootSound("th06_13.mp3");
        level1Song.playLoop();
    }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.
> prepare(): Memanggil metode untuk menempatkan objek pada world.
> level1Song: Memutar musik latar secara berulang (looping).



```
    private void prepare() {
        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        Gatot_StandBy gatot_StandBy = new Gatot_StandBy(90, 130);
        addObject(gatot_StandBy, 114, 480);
        gatot_StandBy.setLocation(111, 487);

        constrainEntityToWorldBounds(gatot_StandBy);

        //timer
        Timer timer = new Timer();
        addObject(timer, 1180, 50);

        //platform
        PlatformTiles platformTiles = new PlatformTiles();
        addObject(platformTiles, 200, 381);

        PlatformTiles platformTiles2 = new PlatformTiles();
        addObject(platformTiles2, 1080, 381);
```

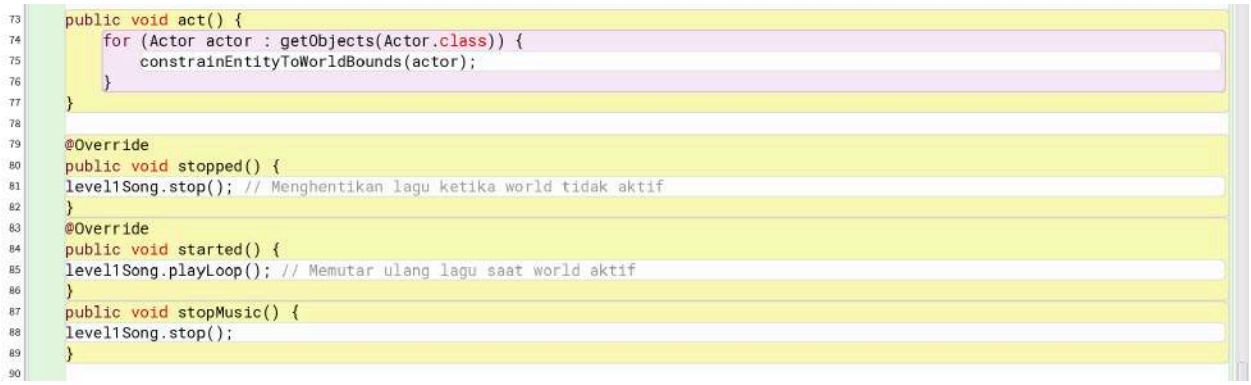


```
        PlatformTiles platformTiles3 = new PlatformTiles();
        addObject(platformTiles3, 482, 243);

        PlatformTiles platformTiles4 = new PlatformTiles();
        addObject(platformTiles4, 633, 243);

        PlatformTiles platformTiles5 = new PlatformTiles();
        addObject(platformTiles5, 784, 243);

        //add heart
        Heart heart1 = new Heart();
        heart1.getImage().scale(100, 100);
        addObject(heart1, 640, 150);

        Heart heart2 = new Heart();
        heart2.getImage().scale(100, 100);
        addObject(heart2, 785, 150);

        Heart heart3 = new Heart();
        heart3.getImage().scale(100, 100);
        addObject(heart3, 480, 150);

        //add boss
        BossMonster bossMonster = new BossMonster();
        addObject(bossMonster,1116,461);
    }
```

Penjelasan :

```
private void prepareLevel() {
        …..;
}
```

> Menambahkan objek yang diperlukan pada world ini sesuai koordinat yang ditentukan.
NOTE : Looping dilakukan untuk menambahkan tile setiap 128 px hingga lebar world tercapai. Tile ini menjadi dasar (ground) di posisi y = 651.



```
public void act() {
    for (Actor actor : getObjects(Actor.class)) {
        constrainEntityToWorldBounds(actor);
    }
}

@Override
public void stopped() {
level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level1Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level1Song.stop();
}
```

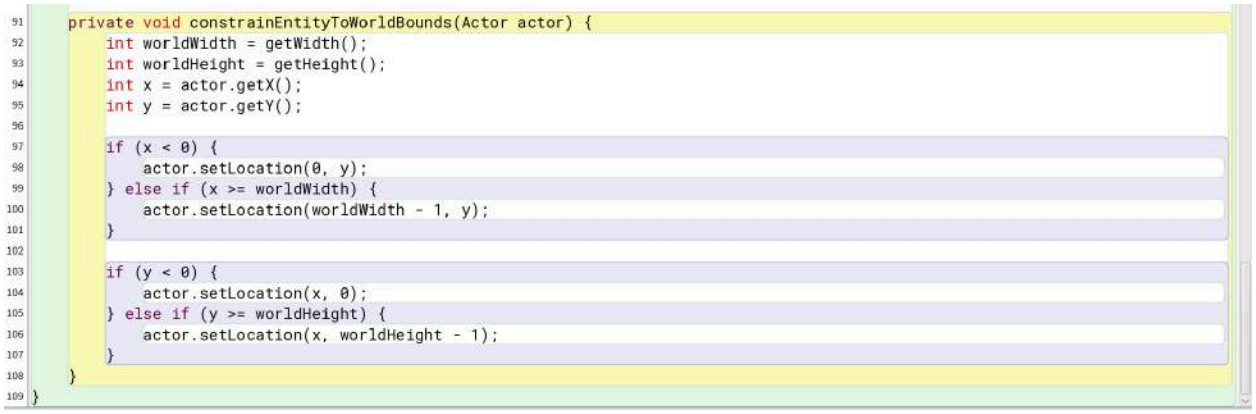
Penjelasan :

```
public void act() {
      for (Actor actor : getObjects(Actor.class)) {
         constrainEntityToWorldBounds(actor);
      }
   }
```

> Metode ini dipanggil setiap frame. Loop memeriksa semua objek di world yang merupakan turunan dari kelas Actor dan memastikan mereka tidak keluar dari batas world.

```
   @Override
   public void stopped() {
   level1Song.stop(); // Menghentikan lagu ketika world tidak aktif
   }
   @Override
   public void started() {
   level1Song.playLoop(); // Memutar ulang lagu saat world aktif
   }
   public void stopMusic() {
   level1Song.stop();
   }
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.



```
    private void constrainEntityToWorldBounds(Actor actor) {
        int worldWidth = getWidth();
        int worldHeight = getHeight();
        int x = actor.getX();
        int y = actor.getY();

        if (x < 0) {
            actor.setLocation(0, y);
        } else if (x >= worldWidth) {
            actor.setLocation(worldWidth - 1, y);
        }

        if (y < 0) {
            actor.setLocation(x, 0);
        } else if (y >= worldHeight) {
            actor.setLocation(x, worldHeight - 1);
        }
    }
}
```

Penjelasan :

```
private void constrainEntityToWorldBounds(Actor actor) {
      int worldWidth = getWidth();
      int worldHeight = getHeight();
      int x = actor.getX();
      int y = actor.getY();

      if (x < 0) {
         actor.setLocation(0, y);
      } else if (x >= worldWidth) {
         actor.setLocation(worldWidth - 1, y);
      }

      if (y < 0) {
         actor.setLocation(x, 0);
      } else if (y >= worldHeight) {
         actor.setLocation(x, worldHeight - 1);
      }
  }
```

> Memastikan setiap actor tidak keluar dari batas width dan height world. Jika posisi x atau y berada di luar batas, posisinya dikembalikan ke batas yang valid.

**BossFight2, BossFight3, BossFight4, BossFight5 memiliki kode Java yang sama seperti BossFight1, perbedaannya hanya terdapat pada :**

1. Musik
    - BossFight2 : level2Song = new GreenfootSound("th06_13.mp3");
    - BossFight3 : level3Song = new GreenfootSound("th06_13.mp3");
    - BossFight4 : level4Song = new GreenfootSound("th06_13.mp3");
    - BossFight5 : level5Song = new GreenfootSound("th06_13.mp3");
2. Penempatan objek

> Objek seperti Heart dan platform diletakkan pada koordinat yang berbeda-beda pada setiap world BossFight sesuai kebutuhannya.

3. Background (Background diset langsung melalui setimage pada setiap world BossFight)
4. Objek boss yang berbeda-beda setiap levelnya

**BossFight2**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class BossFight1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BossFight2 extends BaseWorld {
    private GreenfootSound level2Song;
    public BossFight2() {
        super(1280, 720, 1);
        prepare();

        level2Song = new GreenfootSound("th06_13.mp3");
        level2Song.playLoop();
    }

    private void prepare() {
        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }
```

```
        // Add the Gatot_StandBy actor
        Gatot_StandBy gatot_StandBy = new Gatot_StandBy(90, 130);
        addObject(gatot_StandBy, 114, 480);
        gatot_StandBy.setLocation(111, 487);

        // Constrain the entity's position within the world width
        constrainEntityToWorldBounds(gatot_StandBy);

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        //add platform
        PlatformTiles platformTiles = new PlatformTiles();
        addObject(platformTiles, 200, 381);

        PlatformTiles platformTiles2 = new PlatformTiles();
        addObject(platformTiles2, 1080, 381);

        PlatformTiles platformTiles3 = new PlatformTiles();
        addObject(platformTiles3, 482, 243);

        PlatformTiles platformTiles4 = new PlatformTiles();
        addObject(platformTiles4, 784, 243);

        //add heart
        Heart heart1 = new Heart();
        heart1.getImage().scale(100, 100);
        addObject(heart1, 480, 150);

        Heart heart2 = new Heart();
        heart2.getImage().scale(100, 100);
```

```
        addObject(heart2, 785, 150);

        //add boss
        BossMonster2 bossMonster2 = new BossMonster2();
        addObject(bossMonster2,1080,490);
    }

    public void act() {
        // Ensure any entity added to the world is kept within bounds
        for (Actor actor : getObjects(Actor.class)) {
            constrainEntityToWorldBounds(actor);
        }
    }

    @Override
    public void stopped() {
    level2Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level2Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level2Song.stop();
    }

    private void constrainEntityToWorldBounds(Actor actor) {
        int worldWidth = getWidth();
        int worldHeight = getHeight();
        int x = actor.getX();
        int y = actor.getY();

        // Constrain x position (keep it within the world width)
        if (x < 0) {
            actor.setLocation(0, y);
        } else if (x >= worldWidth) {
            actor.setLocation(worldWidth - 1, y);
        }

        // Constrain y position (if needed, to keep entities within the vertical bounds)
        if (y < 0) {
            actor.setLocation(x, 0);
        } else if (y >= worldHeight) {
            actor.setLocation(x, worldHeight - 1);
        }
    }
}
```

## BossFight3

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class BossFight1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BossFight3 extends BaseWorld {
    private GreenfootSound level3Song;
    public BossFight3() {
        super(1280, 720, 1);
        prepare();

        level3Song = new GreenfootSound("th06_13.mp3");
        level3Song.playLoop();
    }

    private void prepare() {
        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles tile = new Tiles();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        // Add the Gatot_StandBy actor
        Gatot_StandBy gatot_StandBy = new Gatot_StandBy(90, 130);
        addObject(gatot_StandBy, 114, 480);
        gatot_StandBy.setLocation(111, 487);
        // Constrain the entity's position within the world width
        constrainEntityToWorldBounds(gatot_StandBy);

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        //add platform
        PlatformTiles platformTiles = new PlatformTiles();
        addObject(platformTiles, 200, 381);

        PlatformTiles platformTiles2 = new PlatformTiles();
        addObject(platformTiles2, 1080, 381);

        Crate1 crate11 = new Crate1();
        addObject(crate11, 482, 243);
        Crate1 crate12 = new Crate1();
        addObject(crate12, 784, 243);

        //add heart
        Heart heart1 = new Heart();
        heart1.getImage().scale(100, 100);
        addObject(heart1, 480, 150);

        Heart heart2 = new Heart();
        heart2.getImage().scale(100, 100);
        addObject(heart2, 785, 150);

        //add boss
        BossMonster3 bossMonster3 = new BossMonster3();
        addObject(bossMonster3,1080,460);
    }
```

```java
public void act() {
    // Ensure any entity added to the world is kept within bounds
    for (Actor actor : getObjects(Actor.class)) {
        constrainEntityToWorldBounds(actor);
    }
}
```

```java
@Override
public void stopped() {
level3Song.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
level3Song.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
level3Song.stop();
}

private void constrainEntityToWorldBounds(Actor actor) {
    int worldWidth = getWidth();
    int worldHeight = getHeight();
    int x = actor.getX();
    int y = actor.getY();

    // Constrain x position (keep it within the world width)
    if (x < 0) {
        actor.setLocation(0, y);
    } else if (x >= worldWidth) {
        actor.setLocation(worldWidth - 1, y);
    }

    // Constrain y position (if needed, to keep entities within the vertical bounds)
    if (y < 0) {
        actor.setLocation(x, 0);
    } else if (y >= worldHeight) {
        actor.setLocation(x, worldHeight - 1);
    }
}
```

## BossFight4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class BossFight1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BossFight4 extends BaseWorld {
    private GreenfootSound level4Song;
    public BossFight4() {
        super(1280, 720, 1);
        prepare();
        level4Song = new GreenfootSound("th06_13.mp3");
        level4Song.playLoop();
    }

    private void prepare() {
        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles2 tile = new Tiles2();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        // Add the Gatot_StandBy actor
        Gatot_StandBy gatot_StandBy = new Gatot_StandBy(90, 130);
        addObject(gatot_StandBy, 114, 480);
        gatot_StandBy.setLocation(111, 487);
```

```
        // Constrain the entity's position within the world width
        constrainEntityToWorldBounds(gatot_StandBy);

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        Crate1 crate1 = new Crate1();
        addObject(crate1, 200, 381);

        Crate1 crate12 = new Crate1();
        addObject(crate12, 1080, 381);

        Crate1 crate13 = new Crate1();
        addObject(crate13, 482, 243);

        Crate1 crate14 = new Crate1();
        addObject(crate14, 784, 243);

        //add heart
        Heart heart1 = new Heart();
        heart1.getImage().scale(100, 100);
        addObject(heart1, 785, 150);

        Heart heart2 = new Heart();
        heart2.getImage().scale(100, 100);
        addObject(heart2, 480, 150);

        //add boss
        BossMonster4 bossMonster4 = new BossMonster4();
        addObject(bossMonster4,1100,480);
    }
```

```
    public void act() {
        // Ensure any entity added to the world is kept within bounds
        for (Actor actor : getObjects(Actor.class)) {
            constrainEntityToWorldBounds(actor);
        }
    }

    @Override
    public void stopped() {
    level4Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level4Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level4Song.stop();
    }
```

```
    private void constrainEntityToWorldBounds(Actor actor) {
        int worldWidth = getWidth();
        int worldHeight = getHeight();
        int x = actor.getX();
        int y = actor.getY();

        // Constrain x position (keep it within the world width)
        if (x < 0) {
            actor.setLocation(0, y);
        } else if (x >= worldWidth) {
            actor.setLocation(worldWidth - 1, y);
        }

        // Constrain y position (if needed, to keep entities within the vertical bounds)
        if (y < 0) {
            actor.setLocation(x, 0);
        } else if (y >= worldHeight) {
            actor.setLocation(x, worldHeight - 1);
        }
    }
}
```

**BossFight5**

```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class BossFight1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BossFight5 extends BaseWorld {
    private GreenfootSound level5Song;
    public BossFight5() {
        super(1280, 720, 1);
        prepare();
        level5Song = new GreenfootSound("th06_13.mp3");
        level5Song.playLoop();
    }

    private void prepare() {
        int worldWidth = getWidth();
        int tileWidth = 128;
        int yPosition = 651;

        for (int x = 0; x < worldWidth; x += tileWidth) {
            Tiles2 tile = new Tiles2();
            addObject(tile, x + tileWidth / 2, yPosition);
        }

        // Add the Gatot_StandBy actor
        Gatot_StandBy gatot_StandBy = new Gatot_StandBy(90, 130);
        addObject(gatot_StandBy, 114, 480);
        gatot_StandBy.setLocation(111, 487);
        // Constrain the entity's position within the world width
        constrainEntityToWorldBounds(gatot_StandBy);

        //timer
        Timer timer = new Timer();  // Initialize the timer
        addObject(timer, 1180, 50);  // Add the timer to the world

        Crate1 crate1 = new Crate1();
        addObject(crate1, 360, 381);

        Crate1 crate12 = new Crate1();
        addObject(crate12, 900, 381);

        Crate1 crate13 = new Crate1();
        addObject(crate13, 633, 243);

        //add heart
        Heart heart1 = new Heart();
        heart1.getImage().scale(100, 100);
        addObject(heart1, 640, 150);

        //add boss
        BossMonster5 bossMonster5 = new BossMonster5();
        addObject(bossMonster5,1100,480);
    }
```

```
    public void act() {
        // Ensure any entity added to the world is kept within bounds
        for (Actor actor : getObjects(Actor.class)) {
            constrainEntityToWorldBounds(actor);
        }
    }

    @Override
    public void stopped() {
    level5Song.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    level5Song.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    level5Song.stop();
    }
```

```
    private void constrainEntityToWorldBounds(Actor actor) {
        int worldWidth = getWidth();
        int worldHeight = getHeight();
        int x = actor.getX();
        int y = actor.getY();

        // Constrain x position (keep it within the world width)
        if (x < 0) {
            actor.setLocation(0, y);
        } else if (x >= worldWidth) {
            actor.setLocation(worldWidth - 1, y);
        }

        // Constrain y position (if needed, to keep entities within the vertical bounds)
        if (y < 0) {
            actor.setLocation(x, 0);
        } else if (y >= worldHeight) {
            actor.setLocation(x, worldHeight - 1);
        }
    }
}
```

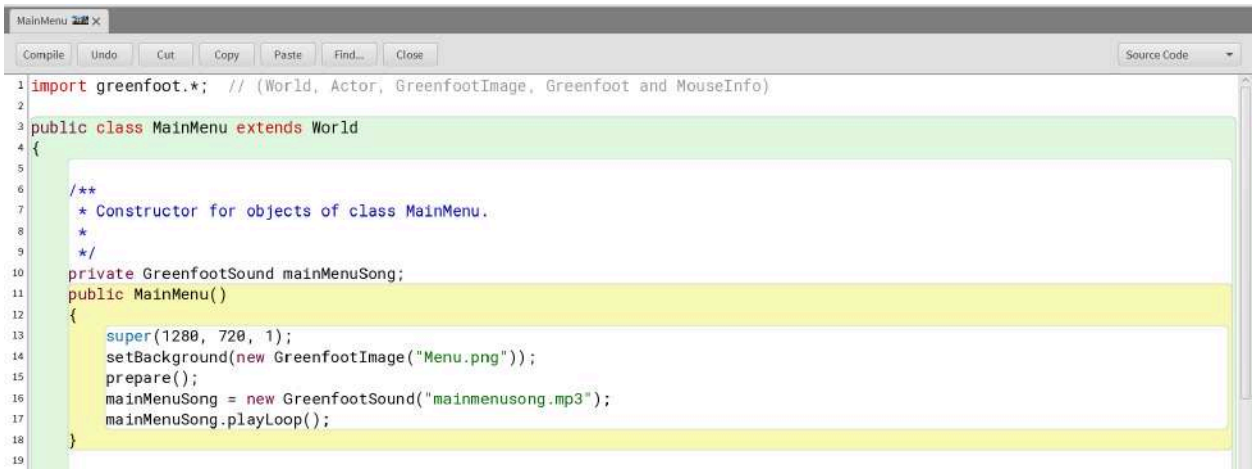
**Setelah pemain melalui world BossFight maka akan lanjut ke Level selanjutnya, apabila sudah berada pada BossFight5 maka player akan pindah ke world victorious lalu Game_Over.**

**Alur Lengkap Level apabila pemain lanjut bermain terus hingga level akhir :**
Level1 > FiLLiT1 > NumQuest1 > Labyrntine1 > BossFight1 > Level2 > FiLLiT2 > NumQuest2 > Labyrntine2 > BossFight2 > Level2 > FiLLiT2 > NumQuest2 > Labyrntine2 > BossFight2 > Level3 > FiLLiT3 > NumQuest3 > Labyrntine3 > BossFight3 > Level4 > FiLLiT4 > NumQuest4 > Labyrntine4 > BossFight4 > Level5 > FiLLiT5 > NumQuest5 > Labyrntine5 > BossFight5 > victorious > Game_Over > historyMenu.

**World Menu (Extends World, berarti tidak menampilkan score, health, dan timer yang ada pada BaseWorld)**

**MainMenu**

World awal yang akan dilihat oleh player apabila membuka game Gatot_Rider.




Penjelasan :

```
private GreenfootSound mainMenuSong;
```

> MainMenuSong: Variabel untuk menyimpan musik yang dimainkan di menu utama secara berulang-ulang.

```
public MainMenu()
    {
        super(1280, 720, 1);
        setBackground(new GreenfootImage("Menu.png"));
        prepare();
        mainMenuSong = new GreenfootSound("mainmenusong.mp3");
        mainMenuSong.playLoop();
    }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.

> setBackground(new GreenfootImage("Menu.png")): Menetapkan latar belakang world dengan gambar Menu.png.

> prepare(): Memanggil metode prepare() untuk menambahkan objek (buttons) interaktif ke menu.

> mainMenuSong = new GreenfootSound("mainmenusong.mp3"): mainMenuSong akan memutar file audio mainmenusong.mp3.

> mainMenuSong.playLoop(): Memainkan musik secara berulang (looping) saat menu utama ditampilkan.

```
    private void prepare() {
        btnPlay btnPlay = new btnPlay();
        btnTutor btnTutor = new btnTutor();
        btnHistory btnHistory = new btnHistory();
        btnAbout btnAbout = new btnAbout();

        addObject(btnPlay, 639, 336);
        addObject(btnTutor, 640, 483);
        addObject(btnHistory, 510, 483);
        addObject(btnAbout, 771, 483);
    }

    public void act() {
    }

    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

Penjelasan :

```
private void prepare() {
        btnPlay btnPlay = new btnPlay();
        btnTutor btnTutor = new btnTutor();
        btnHistory btnHistory = new btnHistory();
        btnAbout btnAbout = new btnAbout();

        addObject(btnPlay, 639, 336);
        addObject(btnTutor, 640, 483);
        addObject(btnHistory, 510, 483);
        addObject(btnAbout, 771, 483);
    }
```

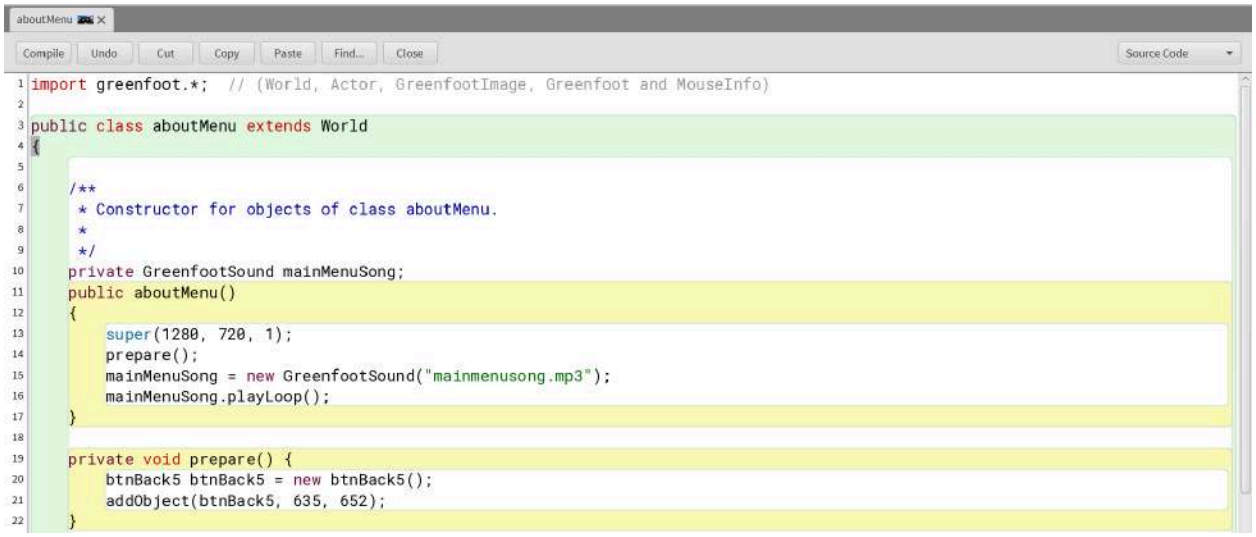
> Menempatkan objek (tombol) pada world sesuai koordinat.

```
   @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.

> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

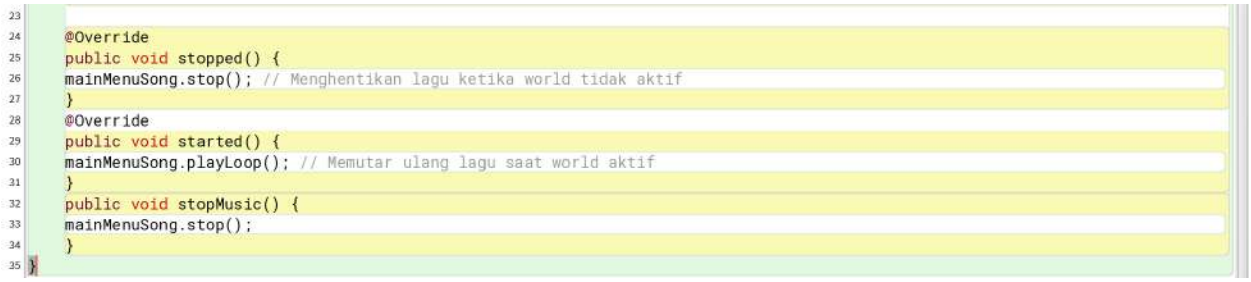
**aboutMenu**
World menu yang menampilkan kredit dari developer game ini.




Penjelasan :
private GreenfootSound mainMenuSong;
> MainMenuSong: Variabel untuk menyimpan musik yang dimainkan di menu utama secara berulang-ulang.

```
public aboutMenu()
   {
      super(1280, 720, 1);
      prepare();
      mainMenuSong = new GreenfootSound("mainmenusong.mp3");
      mainMenuSong.playLoop();
   }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.
> prepare(): Memanggil metode prepare() untuk menambahkan objek (button back) interaktif ke aboutMenu.
> mainMenuSong = new GreenfootSound("mainmenusong.mp3"): mainMenuSong akan memutar file audio mainmenusong.mp3.
> mainMenuSong.playLoop(): Memainkan musik secara berulang (looping) saat menu utama ditampilkan.



```
    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

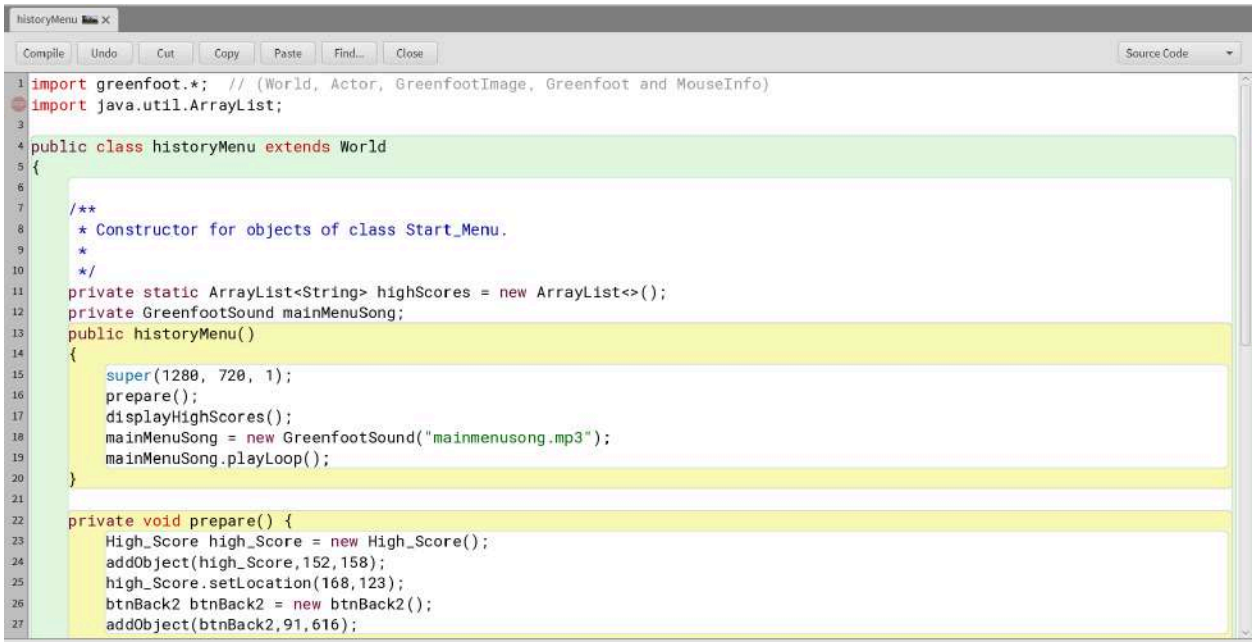
Penjelasan :

```
  @Override
   public void stopped() {
   mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
   }
   @Override
   public void started() {
   mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
   }
   public void stopMusic() {
   mainMenuSong.stop();
   }
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

**historyMenu**

World menu yang menyimpan dan menunjukan High_Score serta Name dan Counter history dari pemain.




Penjelasan :

```
private static ArrayList<String> highScores = new ArrayList<>();
private GreenfootSound mainMenuSong;
```

> highScores: Variabel statis (berbagi data di semua instance) berupa ArrayList untuk menyimpan daftar skor tertinggi dalam format "Name - Counter".
> mainMenuSong: Memutar musik latar selama historyMenu ditampilkan.

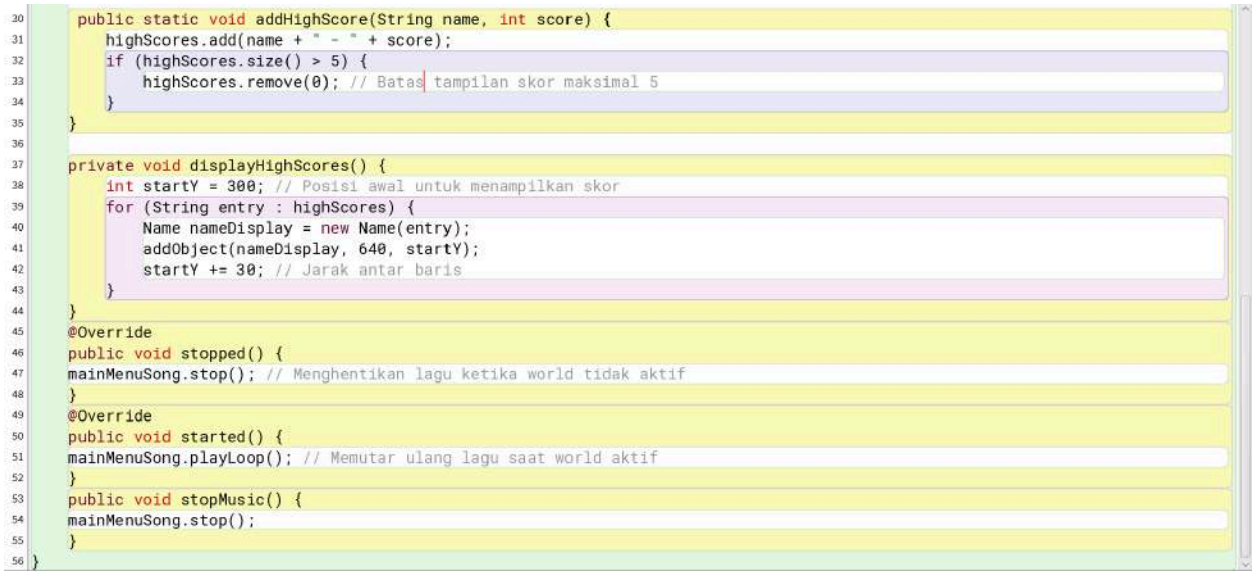
```
public historyMenu()
   {
      super(1280, 720, 1);
      prepare();
      displayHighScores();
      mainMenuSong = new GreenfootSound("mainmenusong.mp3");
      mainMenuSong.playLoop();
   }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.
> prepare(): Memanggil metode prepare() untuk menambahkan objek (button back) interaktif ke aboutMenu.
> displayHighScores(): Memanggil metode untuk menampilkan skor tertinggi pada layar.
> mainMenuSong = new GreenfootSound("mainmenusong.mp3"): mainMenuSong akan memutar file audio mainmenusong.mp3.
> mainMenuSong.playLoop(): Memainkan musik secara berulang (looping) saat menu utama ditampilkan.

```
private void prepare() {
```

```
        High_Score high_Score = new High_Score();
        addObject(high_Score,152,158);
        high_Score.setLocation(168,123);
        btnBack2 btnBack2 = new btnBack2();
        addObject(btnBack2,91,616);
    }
```

> Menempatkan objek seperti High_Score dan btnBack2 pada world.



```
    public static void addHighScore(String name, int score) {
        highScores.add(name + " - " + score);
        if (highScores.size() > 5) {
            highScores.remove(0); // Batas tampilan skor maksimal 5
        }
    }

    private void displayHighScores() {
        int startY = 300; // Posisi awal untuk menampilkan skor
        for (String entry : highScores) {
            Name nameDisplay = new Name(entry);
            addObject(nameDisplay, 640, startY);
            startY += 30; // Jarak antar baris
        }
    }
    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

Penjelasan :

```
public static void addHighScore(String name, int score) {
        highScores.add(name + " - " + score);
        if (highScores.size() > 5) {
            highScores.remove(0); // Batas tampilan skor maksimal 5
        }
    }
```

> Menambahkan skor baru ke daftar highScores dalam format "Name - Counter". Jika jumlah skor melebihi 5, skor paling awal dihapus untuk membatasi daftar hanya 5 skor terakhir.

```
private void displayHighScores() {
        int startY = 300; // Posisi awal untuk menampilkan skor
        for (String entry : highScores) {
            Name nameDisplay = new Name(entry);
            addObject(nameDisplay, 640, startY);
            startY += 30; // Jarak antar baris
        }
    }
```

> Menampilkan setiap skor dari highScores di layar, dengan posisi vertikal mulai dari y = 300.

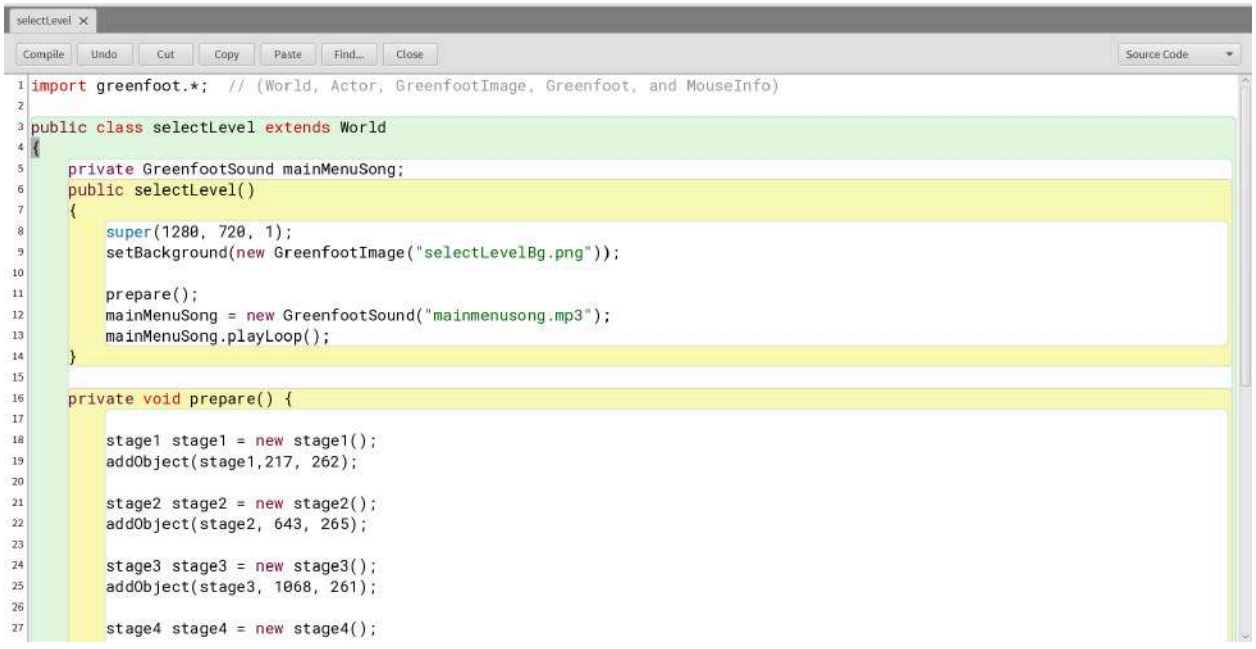
> Setiap skor ditampilkan menggunakan objek Name.
> Skor ditampilkan berurutan secara vertikal, dengan jarak antar baris sebesar 30 piksel.

```
@Override
public void stopped() {
mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
}
@Override
public void started() {
mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
}
public void stopMusic() {
mainMenuSong.stop();
}
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

**selectLevel**

Player mendapatkan opsi untuk memilih darimana ia ingin memulai permainan (bisa memilih level mana yang ingin dimainkan).



```
        addObject(stage4, 361, 494);

        stage5 stage5 = new stage5();
        addObject(stage5, 920, 495);

        btnBack btnBack = new btnBack();
        addObject(btnBack, getWidth() / 2, 620);

    }
    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

Penjelasan :

```
private GreenfootSound mainMenuSong;
```

> mainMenuSong: Memutar musik latar selama selectLevel ditampilkan

```
public selectLevel()
   {
      super(1280, 720, 1);
      setBackground(new GreenfootImage("selectLevelBg.png"));
      prepare();
      mainMenuSong = new GreenfootSound("mainmenusong.mp3");
      mainMenuSong.playLoop();
   }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.
> setBackground(new GreenfootImage("selectLevelBg.png")): Menetapkan latar belakang world dengan gambar Menu.png.
> prepare(): Memanggil metode prepare() untuk menambahkan objek (buttons) interaktif ke menu.
> mainMenuSong = new GreenfootSound("mainmenusong.mp3"): mainMenuSong akan memutar file audio mainmenusong.mp3.
> mainMenuSong.playLoop(): Memainkan musik secara berulang (looping) saat menu utama ditampilkan.

```
private void prepare() {

      stage1 stage1 = new stage1();
      addObject(stage1,217, 262);

      stage2 stage2 = new stage2();
      addObject(stage2, 643, 265);
```

```
        stage3 stage3 = new stage3();
        addObject(stage3, 1068, 261);

        stage4 stage4 = new stage4();
        addObject(stage4, 361, 494);

        stage5 stage5 = new stage5();
        addObject(stage5, 920, 495);

        btnBack btnBack = new btnBack();
        addObject(btnBack, getWidth() / 2, 620);

    }
```

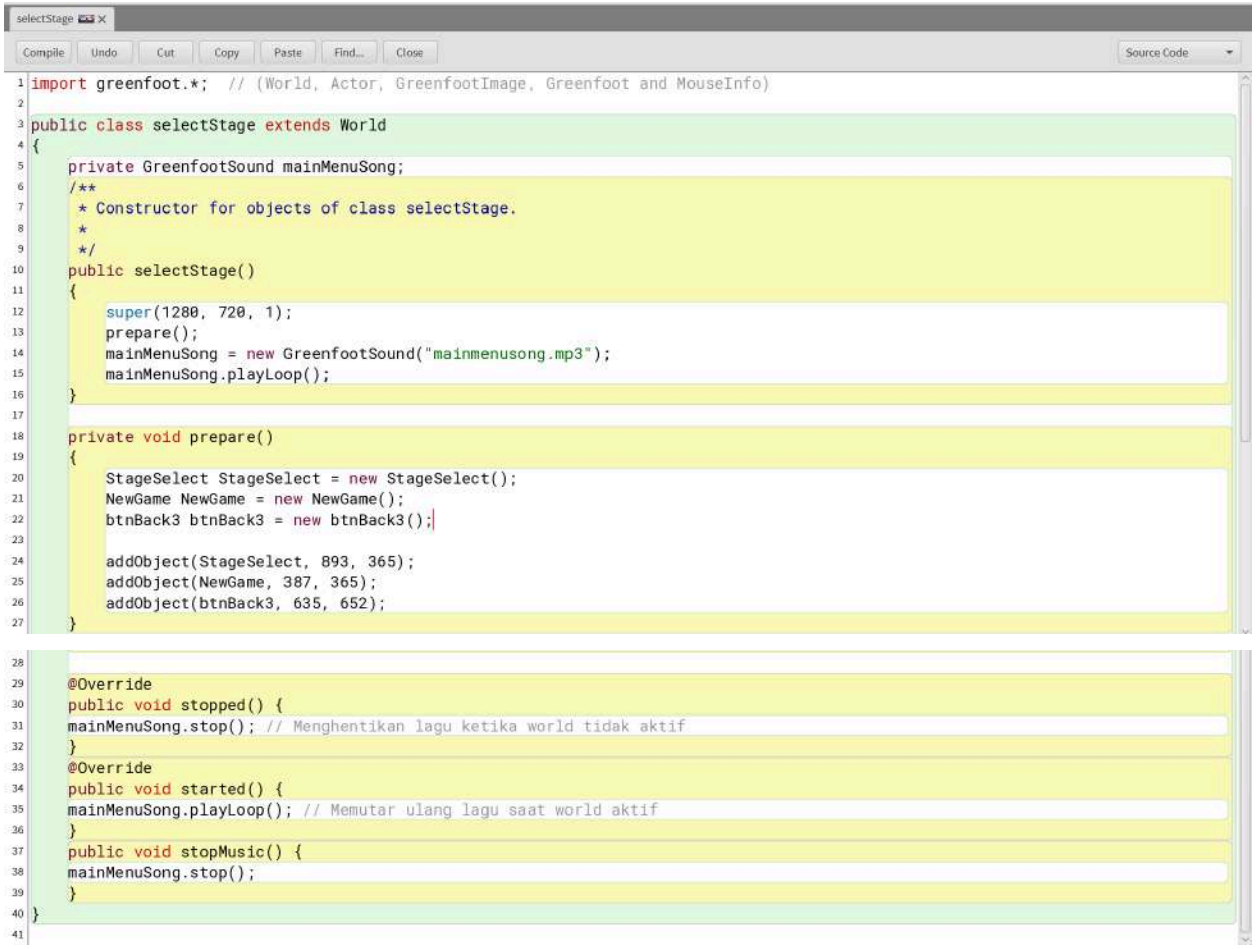
> Menempatkan objek sebagai buttons yang dapat diklik (penjelasan kode pada bagian Actor) sesuai dengan koordinat yang ditentukan.

```
    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

**selectStage**

Pada world ini, pemain diberikan opsi bermain dari Level1 (New Game) atau memilih sendiri mau mulai dari level mana.



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class selectStage extends World
{
    private GreenfootSound mainMenuSong;
    /**
     * Constructor for objects of class selectStage.
     *
     */
    public selectStage()
    {
        super(1280, 720, 1);
        prepare();
        mainMenuSong = new GreenfootSound("mainmenusong.mp3");
        mainMenuSong.playLoop();
    }

    private void prepare()
    {
        StageSelect StageSelect = new StageSelect();
        NewGame NewGame = new NewGame();
        btnBack3 btnBack3 = new btnBack3();

        addObject(StageSelect, 893, 365);
        addObject(NewGame, 387, 365);
        addObject(btnBack3, 635, 652);
    }

    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

Penjelasan :

```
private GreenfootSound mainMenuSong;
```

> mainMenuSong: Memutar musik latar selama selectStage ditampilkan

```
public selectStage()
   {
      super(1280, 720, 1);
      prepare();
      mainMenuSong = new GreenfootSound("mainmenusong.mp3");
      mainMenuSong.playLoop();
   }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.

> prepare(): Memanggil metode prepare() untuk menambahkan objek (buttons) interaktif ke menu.

> mainMenuSong = new GreenfootSound("mainmenusong.mp3"): mainMenuSong akan memutar file audio mainmenusong.mp3.

> mainMenuSong.playLoop(): Memainkan musik secara berulang (looping) saat menu utama ditampilkan.

```
private void prepare()
    {
        StageSelect StageSelect = new StageSelect();
        NewGame NewGame = new NewGame();
        btnBack3 btnBack3 = new btnBack3();

        addObject(StageSelect, 893, 365);
        addObject(NewGame, 387, 365);
        addObject(btnBack3, 635, 652);
    }
```

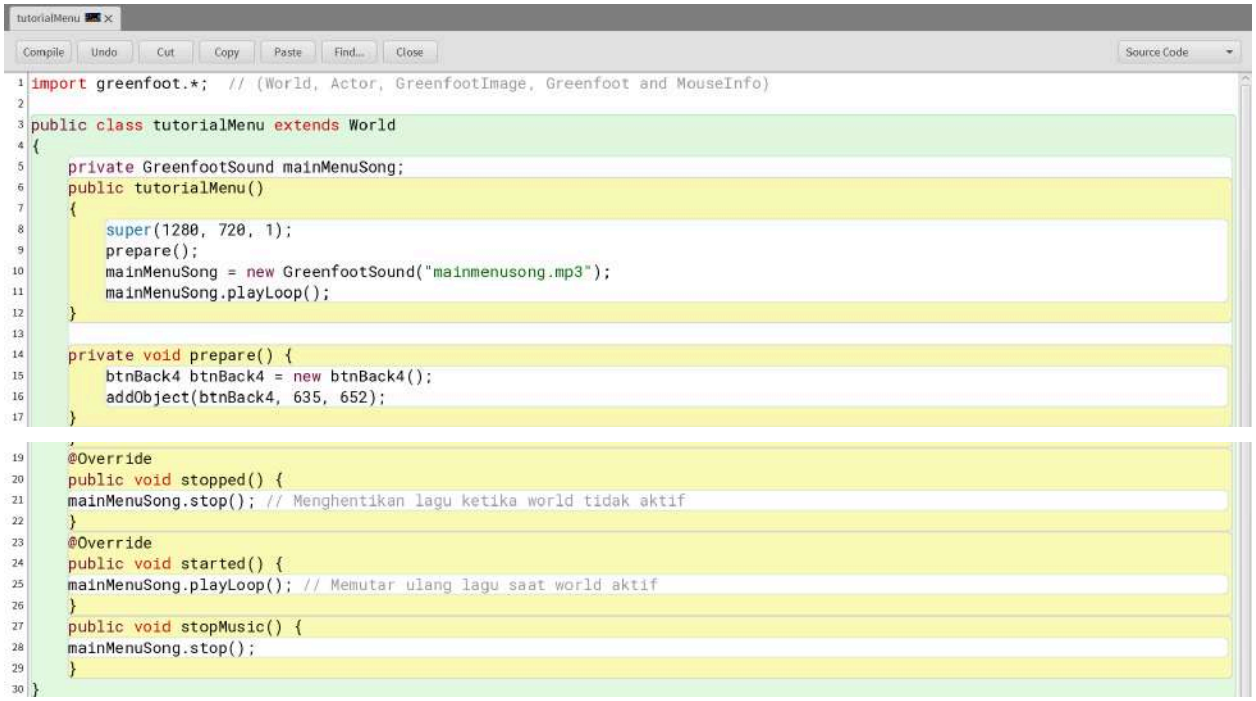
> Menambahkan objek seperti button yang dapat diklik oleh player pada koordinat tertentu.

```
    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
```

> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

**tutorialMenu**

Menu yang berisi instruksi cara bermain game Gatot Rider, menu ini berisi cara kontrol pada game.



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class tutorialMenu extends World
{
    private GreenfootSound mainMenuSong;
    public tutorialMenu()
    {
        super(1280, 720, 1);
        prepare();
        mainMenuSong = new GreenfootSound("mainmenusong.mp3");
        mainMenuSong.playLoop();
    }

    private void prepare() {
        btnBack4 btnBack4 = new btnBack4();
        addObject(btnBack4, 635, 652);
    }

    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
}
```

Penjelasan :

private GreenfootSound mainMenuSong;

> mainMenuSong: Memutar musik latar saat tutorialMenu ditampilkan

```
public tutorialMenu()
    {
        super(1280, 720, 1);
        prepare();
        mainMenuSong = new GreenfootSound("mainmenusong.mp3");
        mainMenuSong.playLoop();
    }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.

> prepare(): Memanggil metode prepare() untuk menambahkan objek (buttons) interaktif ke menu.

> mainMenuSong = new GreenfootSound("mainmenusong.mp3"): mainMenuSong akan memutar file audio mainmenusong.mp3.

> mainMenuSong.playLoop(): Memainkan musik secara berulang (looping) saat menu utama ditampilkan.

```
private void prepare() {
        btnBack4 btnBack4 = new btnBack4();
        addObject(btnBack4, 635, 652);
    }
```
> Menambahkan btnBack4 pada koordinat 635, 652.

```
    @Override
    public void stopped() {
    mainMenuSong.stop(); // Menghentikan lagu ketika world tidak aktif
    }
    @Override
    public void started() {
    mainMenuSong.playLoop(); // Memutar ulang lagu saat world aktif
    }
    public void stopMusic() {
    mainMenuSong.stop();
    }
```
> stopped(): Method menghentikan musik saat tidak di dunia ini.
> started(): Memulai musik kembali ketika dunia dilanjutkan.
> stopMusic(): Metode untuk menghentikan musik secara manual yang akan dipanggil pada objek tertentu sesuai kebutuhan.

**victorious**

Menu yang menunjukan bahwa pemain telah berhasil menyelesaikan seluruh tantangan Gatot Rider.


Penjelasan :

```
private Victory victoryActor;
```
> VictoryActor: Variabel yang menyimpan objek Victory.

```
public victorious(int score) {
        super(1280, 720, 1);
        Greenfoot.playSound("VictoryBGM.wav");
        prepare(score);
    }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1 piksel.
> Greenfoot.playSound("VictoryBGM.wav"): Memutar file audio VictoryBGM.wav sekali, yang berisi musik kemenangan.
> prepare(score): Memanggil metode prepare() dan mengirimkan parameter score untuk menampilkan skor pemain.

```
private void prepare(int score) {
        Victory victoryActor = new Victory(score);
        addObject(victoryActor, getWidth() / 2, getHeight() / 2);
    }
```

> Victory victoryActor = new Victory(score): Membuat objek Victory. Score dikirim sebagai parameter ke konstruktor Victory, untuk menampilkan skor pemain.
> addObject(victoryActor, getWidth() / 2, getHeight() / 2): Menambahkan objek victoryActor ke world, tepat di tengah layar.

**Game_Over**

Menu yang menunjukan bahwa player kalah (kehabisan health) atau bisa juga setelah pemain dibawa ke world victorious, maka akan dibawa ke Game_Over setelahnya untuk input nama agar dapat ditampilkan pada historyMenu Name - Counter nya.


Penjelasan :

```
import java.awt.Color;
```

> Agar bisa menggunakan kelas Color pada java.awt.

```
private GameOver gameOverActor;
```

> Variabel yang menyimpan objek GameOver untuk menampilkan teks yang menunjukkan pesan game over dan skor pemain di layar.

```
public Game_Over(int score) {
        super(1280, 720, 1);
        Greenfoot.playSound("GameOverBGM.wav");
        prepare(score);
    }
```

> super(1280, 720, 1): Mengatur ukuran world menjadi 1280x720 piksel dengan skala 1.
> Greenfoot.playSound("GameOverBGM.wav"): Memutar musik latar "GameOverBGM.wav" yang akan dimainkan sekali.
prepare(score): Memanggil metode prepare() dan mengirimkan parameter score untuk menyiapkan menampilkan skor akhir pemain.

```
private void prepare(int score) {
        gameOverActor = new GameOver(score);
        addObject(gameOverActor, 930, getHeight() / 2);
    }
```

> prepare(int score) membuat objek gameOverActor, yaitu sebuah objek dari kelas GameOver yang menerima skor sebagai parameter. Objek tersebut ditambahkan ke dunia di posisi (930, getHeight() / 2), yaitu secara horizontal di posisi 930 piksel dan vertikal di tengah layar (berdasarkan height).

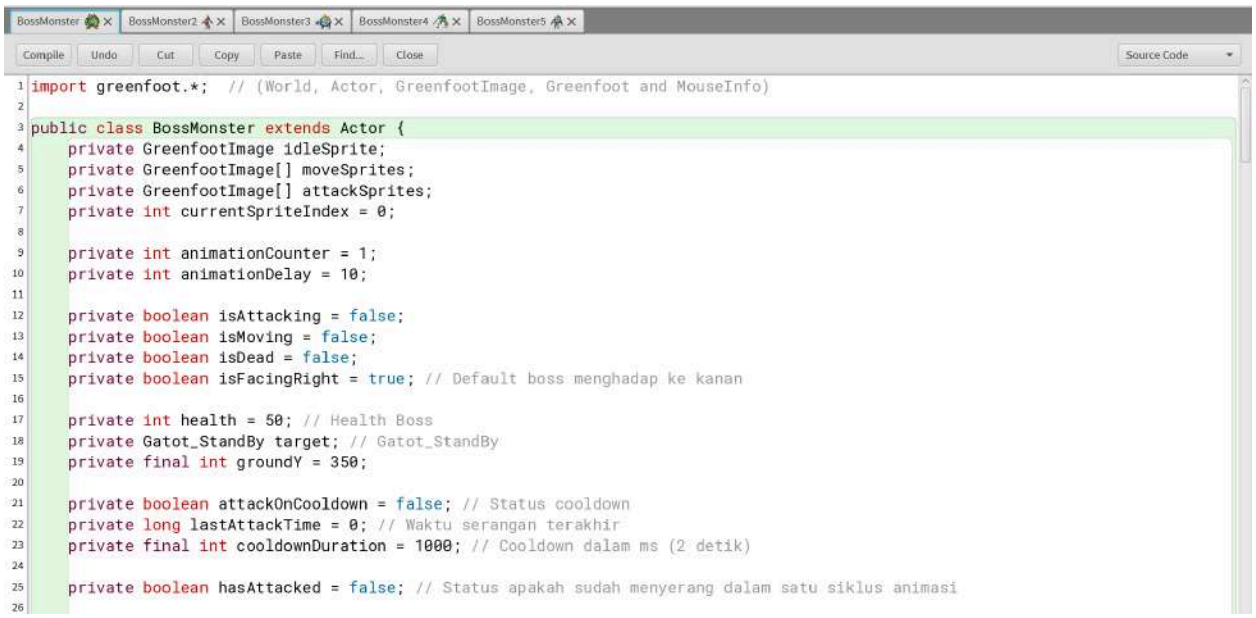
**Alur Lengkap Menu :**
MainMenu >

- historyMenu > MainMenu
- aboutMenu > MainMenu
- tutorialMenu > MainMenu
- selectStage >
    - Level1 > dst.
    - selectLevel >
        - Level1 > dst.
        - Level2 > dst.
        - Level3 > dst.
        - Level4 > dst.
        - Level5 > dst.

Bila HealthBar habis maka :
Game_Over > historyMenu
Bila berhasil menyelesaikan hingga akhit (BossFight5) maka :
Victorious > Game_Over > historyMenu

**Objek (Extends Actor) > Segala objek yang ditempatkan pada world.**
**Beberapa objek pada Gatot Rider diberi methods untuk mendukung berjalannya game ini.**

**BossMonster**




```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BossMonster extends Actor {
    private GreenfootImage idleSprite;
    private GreenfootImage[] moveSprites;
    private GreenfootImage[] attackSprites;
    private int currentSpriteIndex = 0;

    private int animationCounter = 1;
    private int animationDelay = 10;

    private boolean isAttacking = false;
    private boolean isMoving = false;
    private boolean isDead = false;
    private boolean isFacingRight = true; // Default boss menghadap ke kanan

    private int health = 50; // Health Boss
    private Gatot_StandBy target; // Gatot_StandBy
    private final int groundY = 350;

    private boolean attackOnCooldown = false; // Status cooldown
    private long lastAttackTime = 0; // Waktu serangan terakhir
    private final int cooldownDuration = 1000; // Cooldown dalam ms (2 detik)

    private boolean hasAttacked = false; // Status apakah sudah menyerang dalam satu siklus animasi
```

Penjelasan :

```java
private GreenfootImage idleSprite;
private GreenfootImage[] moveSprites;
private GreenfootImage[] attackSprites;
private int currentSpriteIndex = 0;

private int animationCounter = 1;
private int animationDelay = 10;

private boolean isAttacking = false;
private boolean isMoving = false;
private boolean isDead = false;
private boolean isFacingRight = true; // Default boss menghadap ke kanan

private int health = 50; // Health Boss
```

```
    private Gatot_StandBy target; // Gatot_StandBy
    private final int groundY = 350;

    private boolean attackOnCooldown = false; // Status cooldown
    private long lastAttackTime = 0; // Waktu serangan terakhir
    private final int cooldownDuration = 1000; // Cooldown dalam ms (2 detik)

    private boolean hasAttacked = false;
```

> idleSprite, moveSprites, dan attackSprites: Menyimpan sprite untuk animasi diam (idle), bergerak (move), dan menyerang (attack). Sprite ini dimuat dan diubah ukurannya.
> currentSpriteIndex, animationCounter, animationDelay: Mengontrol animasi, termasuk indeks sprite saat ini, penghitung waktu, dan jeda antar frame.
> isAttacking, isMoving, isDead, isFacingRight (Status boss), seperti menyerang, bergerak, mati, dan arah pandang (kanan/kiri).
> health menyimpan nyawa boss, target referensi gatot, groundY agar boss tetap di tanah.
> attackOnCooldown, lastAttackTime, cooldownDuration: Mengatur cooldown antara serangan.
> hasAttacked: Mencegah serangan berulang dalam satu animasi.

```
27      public BossMonster() {
28          // Load idle sprite
29          idleSprite = new GreenfootImage("boss1.png");
30          idleSprite.scale(250, 250); // Perbesar sprite idle
31
32          // Load movement sprites
33          moveSprites = new GreenfootImage[7];
34          for (int i = 0; i < 7; i++) {
35              moveSprites[i] = new GreenfootImage("bossWalk" + (i + 2) + ".png");
36              moveSprites[i].scale(250, 250);
37          }
38
39          // Load attack sprites
40          attackSprites = new GreenfootImage[6];
41          for (int i = 0; i < 6; i++) {
42              attackSprites[i] = new GreenfootImage("bossAttack" + (i + 1) + ".png");
43              attackSprites[i].scale(280, 300);
44          }
45      }
46
47      public void addedToWorld(World world) {
48          target = (Gatot_StandBy) world.getObjects(Gatot_StandBy.class).get(0);
49      }
50
```

Penjelasan :

```
public BossMonster() {
        // Load idle sprite
        idleSprite = new GreenfootImage("boss1.png");
        idleSprite.scale(250, 250); // Perbesar sprite idle

        // Load movement sprites
        moveSprites = new GreenfootImage[7];
        for (int i = 0; i < 7; i++) {
            moveSprites[i] = new GreenfootImage("bossWalk" + (i + 2) + ".png");
```

```
            moveSprites[i].scale(250, 250);
        }

        // Load attack sprites
        attackSprites = new GreenfootImage[6];
        for (int i = 0; i < 6; i++) {
            attackSprites[i] = new GreenfootImage("bossAttack" + (i + 1) + ".png");
            attackSprites[i].scale(280, 300);
        }
    }
```

> Terdapat dua loop untuk memuat sprite animasi gerakan dan serangan. Loop pertama memuat 7 sprite untuk animasi gerakan dari file "bossWalk2.png" hingga "bossWalk8.png" ke dalam array moveSprites, lalu setiap sprite diubah ukurannya menjadi 250x250. Loop kedua memuat 6 sprite untuk animasi serangan dari file "bossAttack1.png" hingga "bossAttack6.png" ke dalam array attackSprites, dengan ukuran masing-masing 280x300.

```
public void addedToWorld(World world) {
        target = (Gatot_StandBy) world.getObjects(Gatot_StandBy.class).get(0);
    }
```

> Mengambil objek Gatot_StandBy di dunia saat boss ditambahkan.

```
51      public void act() {
52          if (!isDead) {
53              stayOnGround();
54              moveToPlayer(); // Move mengikuti Gatot_StandBy
55              checkHitByAttack(); // Check jika kena hit GatotAttackEffect
56          }
57
58          if (isAttacking) {
59              animate(attackSprites);
60          } else if (isMoving) {
61              animate(moveSprites);
62          } else {
63              animateIdle();
64          }
65      }
66
```

Penjelasan :

```
public void act() {
        if (!isDead) {
            stayOnGround();
            moveToPlayer(); // Move mengikuti Gatot_StandBy
            checkHitByAttack(); // Check jika kena hit GatotAttackEffect
        }

        if (isAttacking) {
            animate(attackSprites);
        } else if (isMoving) {
```

```
            animate(moveSprites);
        } else {
            animateIdle();
        }
    }
```

> Boss tetap di tanah: stayOnGround().
> Boss bergerak mendekati pemain: moveToPlayer().
> Mengecek apakah terkena serangan: checkHitByAttack().
> Menentukan animasi dengan kondisi tertentu: Idle, bergerak, atau menyerang.

```
private void animate(GreenfootImage[] sprites) {
    if (animationCounter % animationDelay == 0) {
        currentSpriteIndex = (currentSpriteIndex + 1) % sprites.length; // Hindari out of range
        setImage(sprites[currentSpriteIndex]);

        // Jika animasi selesai saat menyerang, aktifkan cooldown
        if (isAttacking && currentSpriteIndex == sprites.length - 1) {
            isAttacking = false;
            attackOnCooldown = true;
            lastAttackTime = System.currentTimeMillis();
            hasAttacked = false; // Reset status serangan untuk animasi berikutnya
        }
    }
    animationCounter++;
}

private void animateIdle() {
    if (animationCounter % animationDelay == 0) {
        setImage(idleSprite); // Idle hanya menggunakan satu sprite
        currentSpriteIndex = 0; // Reset indeks untuk animasi berikutnya
    }
    animationCounter++;
}
```

Penjelasan :

```
private void animate(GreenfootImage[] sprites) {
        if (animationCounter % animationDelay == 0) {
            currentSpriteIndex = (currentSpriteIndex + 1) % sprites.length; // Hindari out of range
            setImage(sprites[currentSpriteIndex]);

            // Jika animasi selesai saat menyerang, aktifkan cooldown
            if (isAttacking && currentSpriteIndex == sprites.length - 1) {
                isAttacking = false;
                attackOnCooldown = true;
                lastAttackTime = System.currentTimeMillis();
                hasAttacked = false; // Reset status serangan untuk animasi berikutnya
            }
        }
        animationCounter++;
    }
```

> Setiap kali animationCounter mencapai kelipatan animationDelay, indeks gambar saat ini (currentSpriteIndex) ditingkatkan dengan (currentSpriteIndex + 1) % sprites.length untuk

memastikan indeks tetap dalam rentang array. Kemudian, gambar diubah dengan setImage berdasarkan indeks tersebut. Jika animasi sedang dalam mode menyerang (isAttacking) dan telah mencapai gambar terakhir (currentSpriteIndex == sprites.length - 1), serangan dihentikan (isAttacking = false), cooldown diaktifkan (attackOnCooldown = true), waktu serangan terakhir dicatat (lastAttackTime), dan status hasAttacked direset agar animasi siap untuk siklus berikutnya. Akhirnya, animationCounter selalu bertambah untuk melanjutkan ke frame animasi berikutnya.

```
private void animateIdle() {
    if (animationCounter % animationDelay == 0) {
        setImage(idleSprite); // Idle hanya menggunakan satu sprite
        currentSpriteIndex = 0; // Reset indeks untuk animasi berikutnya
    }
    animationCounter++;
}
```

> Jika animationCounter mencapai kelipatan animationDelay, metode akan mengatur gambar menjadi idleSprite karena animasi idle hanya menggunakan satu sprite. Indeks animasi currentSpriteIndex direset ke 0 untuk memastikan transisi yang benar jika animasi lain dimulai. Terakhir, animationCounter selalu meningkat untuk memastikan pengecekan berlanjut di siklus berikutnya. Hal ini memastikan tampilan idle tetap statis dan siap untuk berubah saat kondisi lain terjadi.

```
91  private void moveToPlayer() {
92      if (target != null && !isDead) {
93          int dx = target.getX() - getX();
94          int distance = Math.abs(dx);
95          int speed = 1; // Kecepatan gerak boss
96
97          // Periksa arah dan putar sesuai posisi target
98          if (dx < 0 && !isFacingRight) { // Player di kiri, boss harus menghadap kiri
99              mirrorFacingDirection(true); // Menghadap kiri
100         } else if (dx > 0 && isFacingRight) { // Player di kanan, boss harus menghadap kanan
101             mirrorFacingDirection(false); // Menghadap kanan
102         }
103
104         // Jika jarak cukup dekat, boss menyerang
105         if (distance <= 90) {
106             if (!attackOnCooldown) { // Hanya serang jika tidak dalam cooldown
107                 isAttacking = true;
108                 isMoving = false;
109                 attackPlayer(); // Panggil serangan
110             }
111         } else { // Jika tidak, boss bergerak mendekati player
112             isAttacking = false;
113             isMoving = true;
114
115             // Update posisi boss
116             setLocation(getX() + (dx > 0 ? speed : -speed), getY());
117         }
```

```
        }

        // Reset cooldown jika sudah selesai
        if (attackOnCooldown && (System.currentTimeMillis() - lastAttackTime >= cooldownDuration)) {
            attackOnCooldown = false;
        }
    }

    private void mirrorFacingDirection(boolean faceRight) {
        if (faceRight != isFacingRight) { // Hanya ubah arah jika berbeda
            for (GreenfootImage img : moveSprites) {
                img.mirrorHorizontally(); // Balik semua sprite gerakan
            }
            idleSprite.mirrorHorizontally(); // Balik sprite idle
            for (GreenfootImage img : attackSprites) {
                img.mirrorHorizontally(); // Balik semua sprite serangan
            }
            isFacingRight = faceRight; // Perbarui status arah
        }
    }
```

Penjelasan :

```
private void moveToPlayer() {
    if (target != null && !isDead) {
        int dx = target.getX() - getX();
        int distance = Math.abs(dx);
        int speed = 1; // Kecepatan gerak boss

        // Periksa arah dan putar sesuai posisi target
        if (dx < 0 && !isFacingRight) { // Player di kiri, boss harus menghadap kiri
            mirrorFacingDirection(true); // Menghadap kiri
        } else if (dx > 0 && isFacingRight) { // Player di kanan, boss harus menghadap kanan
            mirrorFacingDirection(false); // Menghadap kanan
        }

        // Jika jarak cukup dekat, boss menyerang
        if (distance <= 90) {
            if (!attackOnCooldown) { // Hanya serang jika tidak dalam cooldown
                isAttacking = true;
                isMoving = false;
                attackPlayer(); // Panggil serangan
            }
        } else { // Jika tidak, boss bergerak mendekati player
            isAttacking = false;
            isMoving = true;

            // Update posisi boss
            setLocation(getX() + (dx > 0 ? speed : -speed), getY());
        }
    }
```

```
        // Reset cooldown jika sudah selesai
                if (attackOnCooldown && (System.currentTimeMillis() - lastAttackTime >= cooldownDuration)) {
            attackOnCooldown = false;
        }
    }
```

> Metode moveToPlayer mengatur pergerakan dan serangan BossMonster terhadap gatot (target). Pertama, dihitung jarak horizontal (dx) antara posisi boss dan target, serta arah yang harus dihadapinya. Jika target berada di kiri dan boss belum menghadap kiri, atau sebaliknya, metode mirrorFacingDirection dipanggil untuk membalikkan arah sprite. Jika jarak boss ke pemain kurang dari atau sama dengan 90 piksel, dan cooldown serangan tidak aktif, boss akan masuk mode menyerang dengan attackPlayer, menghentikan pergerakan sementara. Jika jaraknya lebih jauh, boss bergerak mendekati target dengan kecepatan tetap. Terakhir, jika cooldown aktif dan waktunya selesai, cooldown akan di-reset sehingga boss bisa menyerang kembali. Logika ini memastikan boss bergerak dan menyerang secara dinamis sesuai posisi pemain.

```
private void mirrorFacingDirection(boolean faceRight) {
        if (faceRight != isFacingRight) { // Hanya ubah arah jika berbeda
            for (GreenfootImage img : moveSprites) {
                img.mirrorHorizontally(); // Balik semua sprite gerakan
            }
            idleSprite.mirrorHorizontally(); // Balik sprite idle
            for (GreenfootImage img : attackSprites) {
                img.mirrorHorizontally(); // Balik semua sprite serangan
            }
            isFacingRight = faceRight; // Perbarui status arah
        }
    }
```

> mirrorFacingDirection mengatur arah sprite BossMonster (menghadap kanan atau kiri) dengan membalik gambar sprite secara horizontal. Logika dimulai dengan memeriksa apakah arah yang diminta (faceRight) berbeda dengan arah saat ini (isFacingRight). Jika ya, metode ini menggunakan loop untuk membalik gambar di array moveSprites dan attackSprites, serta membalik sprite idle (idleSprite) dengan memanggil mirrorHorizontally(). Setelah itu, variabel isFacingRight diperbarui untuk mencerminkan arah baru. Metode ini memastikan animasi boss selalu konsisten dengan arah pergerakannya.

```
139     private void checkHitByAttack() {
140         // Check boss hit GatotAttackEffect
141         GatotAttackEffect attack = (GatotAttackEffect) getOneIntersectingObject(GatotAttackEffect.class);
142         if (attack != null) {
143             health -= 1; // Decrease health
144             BaseWorld world = (BaseWorld) getWorld();
145             if (world != null) {
146                 world.removeObject(attack); // Remove attack object
147             }
148 
149             if (health <= 0) {
150                 world.getCounter().addScore(50);
151                 isDead = true; // boss dead
152                 if (world != null) {
153                     world.getCounter().addScore(10); // Bonus score
154                     if (world instanceof BossFight1) {
155                         ((BossFight1) world).stopMusic(); // Stop music
156                     }
157                 }
158                 getWorld().removeObject(this); // Remove boss from the world
159                 Greenfoot.setWorld(new Level2()); // Pindah ke world level2
160             }
161         }
162     }
```

Penjelasan :

```
private void checkHitByAttack() {
    // Check boss hit GatotAttackEffect
                                    GatotAttackEffect attack = (GatotAttackEffect)
getOneIntersectingObject(GatotAttackEffect.class);
    if (attack != null) {
        health -= 1; // Decrease health
        BaseWorld world = (BaseWorld) getWorld();
        if (world != null) {
            world.removeObject(attack); // Remove attack object
        }

        if (health <= 0) {
            world.getCounter().addScore(50);
            isDead = true; // boss dead
            if (world != null) {
                world.getCounter().addScore(10); // Bonus score
                if (world instanceof BossFight1) {
                    ((BossFight1) world).stopMusic(); // Stop music
                }
            }
            getWorld().removeObject(this); // Remove boss from the world
            Greenfoot.setWorld(new Level2()); // Pindah ke world level2
        }
    }
}
```

> checkHitByAttack memeriksa apakah boss terkena serangan dari objek GatotAttackEffect. Menggunakan getOneIntersectingObject untuk mendeteksi jika ada serangan yang mengenai

boss. Jika ada, maka health boss berkurang sebesar 1. Selanjutnya, metode menghapus objek serangan dari dunia menggunakan removeObject untuk membersihkan efek visual. Jika health boss mencapai nol atau kurang (health <= 0), boss dianggap mati (isDead = true). Pada kondisi ini, skor pemain bertambah sebanyak 50 poin, dan bonus tambahan 10 poin diberikan. Jika dunia tempat boss berada adalah instance dari BossFight1, musik pada world ini dihentikan dengan stopMusic(). Objek boss dihapus dari dunia, dan pemain dipindahkan ke level berikutnya (Level2). Metode ini mengelola logika serangan, kesehatan, dan transisi antar-level.

```
163
164     private void stayOnGround() {
165         if (getY() < groundY) {
166             setLocation(getX(), groundY);
167         }
168     }
169
170     private void attackPlayer() {
171         if (isAttacking && !hasAttacked) { // menyerang satu kali per animasi
172             Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
173             if (player != null) {
174                 BaseWorld world = (BaseWorld) getWorld();
175                 if (world != null) {
176                     HealthBar healthBar = world.getHealthBar();
177                     healthBar.decreaseLife(); // Kurangi nyawa pemain hanya satu
178                     hasAttacked = true; // Tandai bahwa boss telah menyerang
179                 }
180             }
181         }
182
183         if (currentSpriteIndex == attackSprites.length - 1) {
184             hasAttacked = false; // menyerang lagi di animasi berikutnya
185         }
186     }
187 }
```

Penjelasan :

```
private void stayOnGround() {
      if (getY() < groundY) {
         setLocation(getX(), groundY);
      }
   }
```

> Memastikan boss tetap di groundY.

```
private void attackPlayer() {
      if (isAttacking && !hasAttacked) { // menyerang satu kali per animasi
                                          Gatot_StandBy player = (Gatot_StandBy)
getOneIntersectingObject(Gatot_StandBy.class);
         if (player != null) {
            BaseWorld world = (BaseWorld) getWorld();
            if (world != null) {
               HealthBar healthBar = world.getHealthBar();
               healthBar.decreaseLife(); // Kurangi nyawa pemain hanya satu
               hasAttacked = true; // Tandai bahwa boss telah menyerang
            }
```

```
            }
        }

        if (currentSpriteIndex == attackSprites.length - 1) {
            hasAttacked = false; // menyerang lagi di animasi berikutnya
        }
    }
```

> attackPlayer mengatur serangan boss terhadap pemain. Metode memeriksa apakah boss sedang dalam status menyerang (isAttacking) dan belum menyerang sebelumnya selama animasi serangan (!hasAttacked). Jika kondisi terpenuhi, ia mendeteksi apakah objek Gatot_StandBy (pemain) berada dalam jangkauan serangan menggunakan getOneIntersectingObject. Jika pemain terdeteksi, metode mengambil referensi dunia saat ini dan memanggil decreaseLife dari objek HealthBar untuk mengurangi nyawa pemain sebanyak satu. Kemudian, variabel hasAttacked diatur menjadi true untuk mencegah serangan berulang selama animasi yang sama. Saat animasi serangan mencapai frame terakhir (currentSpriteIndex == attackSprites.length - 1), hasAttacked diatur kembali menjadi false untuk memungkinkan boss menyerang lagi pada siklus animasi berikutnya.

**BossMonster2, BossMonster3, BossMonster4, BossMonster5 memiliki kode Java yang sama seperti BossMonster1, perbedaannya hanya terdapat pada :**

1. Health Boss (Semakin tinggi level, semakin sulit mengalahkan boss karena Healthnya semakin banyak)
    - BossFight2 : private int health = 65;
    - BossFight3 : private int health = 80;
    - BossFight4 : private int health = 95;
    - BossFight5 : private int health = 130;
2. Speed bergerak Boss
    - BossFight2 : int speed = 2;
    - BossFight3 : int speed = 1;
    - BossFight4 : int speed = 3;
    - BossFight5 : int speed = 2;
3. Sprite Boss yang berbeda-beda gambarnya untuk menambahkan unsur variasi pada tampilan setiap Boss.

## BossMonster2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BossMonster2 extends Actor {
    private GreenfootImage idleSprite;
    private GreenfootImage[] moveSprites;
    private GreenfootImage[] attackSprites;
    private int currentSpriteIndex = 0;

    private int animationCounter = 1;
    private int animationDelay = 10;

    private boolean isAttacking = false;
    private boolean isMoving = false;
    private boolean isDead = false;
    public boolean isFacingRight = true; // Default boss menghadap ke kanan

    private int health = 65; // Health Boss
    private Gatot_StandBy target; // Reference to Gatot_StandBy
    private final int groundY = 350;

    private boolean attackOnCooldown = false; // Status cooldown
    private long lastAttackTime = 0; // Waktu serangan terakhir
    private final int cooldownDuration = 1000; // Cooldown dalam milidetik (2 detik)

    private boolean hasAttacked = false; // Status apakah sudah menyerang dalam satu siklus animasi
```

```
    public BossMonster2() {
        // Load idle sprite
        idleSprite = new GreenfootImage("boss2idle1.png");
        idleSprite.scale(180, 180);

        // Load movement sprites
        moveSprites = new GreenfootImage[4];
        for (int i = 0; i < 4; i++) {
            moveSprites[i] = new GreenfootImage("boss2idle" + (i + 2) + ".png");
            moveSprites[i].scale(180, 180); // Memanggil scale pada elemen individual
        }

        // Load attack sprites
        attackSprites = new GreenfootImage[5];
        for (int i = 0; i < 5; i++) {
            attackSprites[i] = new GreenfootImage("boss2attack" + (i + 1) + ".png");
            attackSprites[i].scale(180, 180); // Memanggil scale pada elemen individual
        }
    }


    public void addedToWorld(World world) {
        // Get the Gatot_StandBy object
        target = (Gatot_StandBy) world.getObjects(Gatot_StandBy.class).get(0);
    }
```

```
public void act() {
    if (!isDead) {
        stayOnGround(); // Bos tetap berada di atas tanah
        moveToPlayer(); // Move towards Gatot_StandBy
        checkHitByAttack(); // Check if hit by GatotAttackEffect
    }

    // Handle animation
    if (isAttacking) {
        animate(attackSprites);
    } else if (isMoving) {
        animate(moveSprites);
    } else {
        animateIdle();
    }
}

private void animate(GreenfootImage[] sprites) {
    if (animationCounter % animationDelay == 0) {
        currentSpriteIndex = (currentSpriteIndex + 1) % sprites.length; // Hindari out of range
        setImage(sprites[currentSpriteIndex]);

        // Jika animasi selesai saat menyerang, aktifkan cooldown
        if (isAttacking && currentSpriteIndex == sprites.length - 1) {
            isAttacking = false;
            attackOnCooldown = true;
            lastAttackTime = System.currentTimeMillis();
            hasAttacked = false; // Reset status serangan untuk animasi berikutnya
        }
    }
    animationCounter++;
}
```

```
private void animateIdle() {
    if (animationCounter % animationDelay == 0) {
        setImage(idleSprite); // Idle hanya menggunakan satu sprite
        currentSpriteIndex = 0; // Reset indeks untuk animasi berikutnya
    }
    animationCounter++;
}

private void moveToPlayer() {
    if (target != null && !isDead) {
        int dx = target.getX() - getX(); // Jarak horizontal antara boss dan player
        int distance = Math.abs(dx); // Jarak absolut
        int speed = 2; // Kecepatan gerak boss

        // Periksa arah dan putar sesuai posisi target
        if (dx < 0 && !isFacingRight) { // Player di kiri, boss harus menghadap kanan
            mirrorFacingDirection(true); // Menghadap kanan
        } else if (dx > 0 && isFacingRight) { // Player di kanan, boss harus menghadap kiri
            mirrorFacingDirection(false); // Menghadap kiri
        }

        // Jika jarak cukup dekat, boss menyerang
        if (distance <= 90) {
            if (!attackOnCooldown) { // Hanya serang jika tidak dalam cooldown
                isAttacking = true;
                isMoving = false;
                attackPlayer(); // Panggil logika serangan
            }
        } else { // Jika tidak, boss bergerak mendekati player
            isAttacking = false;
            isMoving = true;
```

```
            // Update posisi boss
            setLocation(getX() + (dx > 0 ? speed : -speed), getY());
        }
    }

    // Reset cooldown jika sudah selesai
    if (attackOnCooldown && (System.currentTimeMillis() - lastAttackTime >= cooldownDuration)) {
        attackOnCooldown = false;
    }
}

private void mirrorFacingDirection(boolean faceRight) {
    if (faceRight != isFacingRight) { // Hanya ubah arah jika berbeda
        for (GreenfootImage img : moveSprites) {
            img.mirrorHorizontally(); // Balik semua sprite gerakan
        }
        idleSprite.mirrorHorizontally(); // Balik sprite idle
        for (GreenfootImage img : attackSprites) {
            img.mirrorHorizontally(); // Balik semua sprite serangan
        }
        isFacingRight = faceRight; // Perbarui status arah
    }
}

private void checkHitByAttack() {
    // Check if boss is hit by GatotAttackEffect
    GatotAttackEffect attack = (GatotAttackEffect) getOneIntersectingObject(GatotAttackEffect.class);
    if (attack != null) {
        health -= 1; // Decrease health
        BaseWorld world = (BaseWorld) getWorld();
        if (world != null) {
            world.removeObject(attack); // Remove the attack object
        }

        if (health <= 0) {
            world.getCounter().addScore(50);
            isDead = true; // Set boss as dead
            if (world != null) {
                world.getCounter().addScore(10); // Bonus score for defeating boss
                if (world instanceof BossFight2) {
                    ((BossFight2) world).stopMusic(); // Stop music
                }
            }
            getWorld().removeObject(this); // Remove boss from the world
            Greenfoot.setWorld(new Level3());
        }
    }
}

private void stayOnGround() {
    if (getY() < groundY) {
        setLocation(getX(), groundY);
    }
}

private void attackPlayer() {
    if (isAttacking && !hasAttacked) { // Pastikan hanya menyerang satu kali per animasi
        Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
        if (player != null) {
            BaseWorld world = (BaseWorld) getWorld();
            if (world != null) {
                HealthBar healthBar = world.getHealthBar();
                healthBar.decreaseLife(); // Kurangi nyawa pemain hanya satu kali
                hasAttacked = true; // Tandai bahwa boss telah menyerang
```

```
            }
        }
    }

    // Reset hasAttacked di akhir animasi serangan
    if (currentSpriteIndex == attackSprites.length - 1) {
        hasAttacked = false; // Siap untuk menyerang lagi di animasi berikutnya
    }
  }
}
```

## BossMonster3

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BossMonster3 extends Actor {
    private GreenfootImage idleSprite;
    private GreenfootImage[] moveSprites;
    private GreenfootImage[] attackSprites;
    private int currentSpriteIndex = 0;

    private int animationCounter = 1;
    private int animationDelay = 10;

    private boolean isAttacking = false;
    private boolean isMoving = false;
    private boolean isDead = false;
    public boolean isFacingRight = true; // Default boss menghadap ke kanan

    private int health = 80; // Health Boss
    private Gatot_StandBy target; // Reference to Gatot_StandBy
    private final int groundY = 350;

    private boolean attackOnCooldown = false; // Status cooldown
    private long lastAttackTime = 0; // Waktu serangan terakhir
    private final int cooldownDuration = 1000; // Cooldown dalam milidetik (2 detik)

    private boolean hasAttacked = false; // Status apakah sudah menyerang dalam satu siklus animasi
```

```
    public BossMonster3() {
        // Load idle sprite
        idleSprite = new GreenfootImage("boss3Idle.png");
        idleSprite.scale(250, 250);
        // Load movement sprites
        moveSprites = new GreenfootImage[7];
        for (int i = 0; i < 7; i++) {
            moveSprites[i] = new GreenfootImage("boss3move" + (i + 2) + ".png");
            moveSprites[i].scale(250, 250);
        }

        // Load attack sprites
        attackSprites = new GreenfootImage[4];
        for (int i = 0; i < 4; i++) {
            attackSprites[i] = new GreenfootImage("boss3Attack" + (i + 1) + ".png");
            attackSprites[i].scale(330, 250);
        }
    }

    public void addedToWorld(World world) {
        // Get the Gatot_StandBy object
        target = (Gatot_StandBy) world.getObjects(Gatot_StandBy.class).get(0);
    }
```

```
public void act() {
    if (!isDead) {
        stayOnGround(); // Bos tetap berada di atas tanah
        moveToPlayer(); // Move towards Gatot_StandBy
        checkHitByAttack(); // Check if hit by GatotAttackEffect
    }

    // Handle animation
    if (isAttacking) {
        animate(attackSprites);
    } else if (isMoving) {
        animate(moveSprites);
    } else {
        animateIdle();
    }
}

private void animate(GreenfootImage[] sprites) {
    if (animationCounter % animationDelay == 0) {
        currentSpriteIndex = (currentSpriteIndex + 1) % sprites.length; // Hindari out of range
        setImage(sprites[currentSpriteIndex]);

        // Jika animasi selesai saat menyerang, aktifkan cooldown
        if (isAttacking && currentSpriteIndex == sprites.length - 1) {
            isAttacking = false;
            attackOnCooldown = true;
            lastAttackTime = System.currentTimeMillis();
            hasAttacked = false; // Reset status serangan untuk animasi berikutnya
        }
    }
    animationCounter++;
}
```

```
private void animateIdle() {
    if (animationCounter % animationDelay == 0) {
        setImage(idleSprite); // Idle hanya menggunakan satu sprite
        currentSpriteIndex = 0; // Reset indeks untuk animasi berikutnya
    }
    animationCounter++;
}

private void moveToPlayer() {
    if (target != null && !isDead) {
        int dx = target.getX() - getX(); // Jarak horizontal antara boss dan player
        int distance = Math.abs(dx); // Jarak absolut
        int speed = 1; // Kecepatan gerak boss

        // Periksa arah dan putar sesuai posisi target
        if (dx < 0 && !isFacingRight) { // Player di kiri, boss harus menghadap kanan
            mirrorFacingDirection(true); // Menghadap kanan
        } else if (dx > 0 && isFacingRight) { // Player di kanan, boss harus menghadap kiri
            mirrorFacingDirection(false); // Menghadap kiri
        }

        // Jika jarak cukup dekat, boss menyerang
        if (distance <= 90) {
            if (!attackOnCooldown) { // Hanya serang jika tidak dalam cooldown
                isAttacking = true;
                isMoving = false;
                attackPlayer(); // Panggil logika serangan
            }
        } else { // Jika tidak, boss bergerak mendekati player
            isAttacking = false;
            isMoving = true;
```

```
            // Update posisi boss
            setLocation(getX() + (dx > 0 ? speed : -speed), getY());
        }
    }

    // Reset cooldown jika sudah selesai
    if (attackOnCooldown && (System.currentTimeMillis() - lastAttackTime >= cooldownDuration)) {
        attackOnCooldown = false;
    }
}

private void mirrorFacingDirection(boolean faceRight) {
    if (faceRight != isFacingRight) { // Hanya ubah arah jika berbeda
        for (GreenfootImage img : moveSprites) {
            img.mirrorHorizontally(); // Balik semua sprite gerakan
        }
        idleSprite.mirrorHorizontally(); // Balik sprite idle
        for (GreenfootImage img : attackSprites) {
            img.mirrorHorizontally(); // Balik semua sprite serangan
        }
        isFacingRight = faceRight; // Perbarui status arah
    }
}
```

```
private void checkHitByAttack() {
    // Check if boss is hit by GatotAttackEffect
    GatotAttackEffect attack = (GatotAttackEffect) getOneIntersectingObject(GatotAttackEffect.class);
    if (attack != null) {
        health -= 1; // Decrease health
        BaseWorld world = (BaseWorld) getWorld();
        if (world != null) {
            world.removeObject(attack); // Remove the attack object
        }

        if (health <= 0) {
            world.getCounter().addScore(50);
            isDead = true; // Set boss as dead
            if (world != null) {
                world.getCounter().addScore(10); // Bonus score for defeating boss
                if (world instanceof BossFight3) {
                    ((BossFight3) world).stopMusic(); // Stop music
                }
            }
            getWorld().removeObject(this); // Remove boss from the world
            Greenfoot.setWorld(new Level4());
        }
    }
}
```

```
    private void stayOnGround() {
        if (getY() < groundY) {
            setLocation(getX(), groundY);
        }
    }

    private void attackPlayer() {
        if (isAttacking && !hasAttacked) { // Pastikan hanya menyerang satu kali per animasi
            Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
            if (player != null) {
                BaseWorld world = (BaseWorld) getWorld();
                if (world != null) {
                    HealthBar healthBar = world.getHealthBar();
                    healthBar.decreaseLife(); // Kurangi nyawa pemain hanya satu kali
                    hasAttacked = true; // Tandai bahwa boss telah menyerang
                }
            }

            // Reset hasAttacked di akhir animasi serangan
            if (currentSpriteIndex == attackSprites.length - 1) {
                hasAttacked = false; // Siap untuk menyerang lagi di animasi berikutnya
            }
        }
    }
}
```

## BossMonster4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BossMonster4 extends Actor {
    private GreenfootImage idleSprite;
    private GreenfootImage moveSprites;
    private GreenfootImage[] attackSprites;
    private int currentSpriteIndex = 0;

    private int animationCounter = 1;
    private int animationDelay = 10;

    private boolean isAttacking = false;
    private boolean isMoving = false;
    private boolean isDead = false;
    public boolean isFacingRight = true; // Default boss menghadap ke kanan

    private int health = 95; // Health Boss
    private Gatot_StandBy target; // Reference to Gatot_StandBy
    private final int groundY = 350;

    private boolean attackOnCooldown = false; // Status cooldown
    private long lastAttackTime = 0; // Waktu serangan terakhir
    private final int cooldownDuration = 1000; // Cooldown dalam milidetik (2 detik)

    private boolean hasAttacked = false; // Status apakah sudah menyerang dalam satu siklus animasi
```

```
public BossMonster4() {
    // Load idle sprite
    idleSprite = new GreenfootImage("boss4idle.png");
    idleSprite.scale(180, 180);

    // Load movement sprite
    moveSprites = new GreenfootImage("boss4move.png");
    moveSprites.scale(180, 180);

    // Load attack sprites
    attackSprites = new GreenfootImage[4];
    for (int i = 0; i < 4; i++) {
        attackSprites[i] = new GreenfootImage("boss4attack" + (i + 1) + ".png");
        attackSprites[i].scale(180, 180);
    }
}

public void addedToWorld(World world) {
    // Get the Gatot_StandBy object
    target = (Gatot_StandBy) world.getObjects(Gatot_StandBy.class).get(0);
}

public void act() {
    if (!isDead) {
        stayOnGround(); // Bos tetap berada di atas tanah
        moveToPlayer(); // Move towards Gatot_StandBy
        checkHitByAttack(); // Check if hit by GatotAttackEffect
    }

    // Handle animation
    if (isAttacking) {
        animate(attackSprites);
    } else if (isMoving) {
        setImage(moveSprites); // Set sprite gerakan saat bergerak
    } else {
        animateIdle(); // Set sprite idle saat diam
    }
}

private void animate(GreenfootImage[] sprites) {
    if (animationCounter % animationDelay == 0) {
        currentSpriteIndex = (currentSpriteIndex + 1) % sprites.length; // Hindari out of range
        setImage(sprites[currentSpriteIndex]);

        // Jika animasi selesai saat menyerang, aktifkan cooldown
        if (isAttacking && currentSpriteIndex == sprites.length - 1) {
            isAttacking = false;
            attackOnCooldown = true;
            lastAttackTime = System.currentTimeMillis();
            hasAttacked = false; // Reset status serangan untuk animasi berikutnya
        }
    }
    animationCounter++;
}
```

```
private void animateIdle() {
    if (animationCounter % animationDelay == 0) {
        setImage(idleSprite); // Idle hanya menggunakan satu sprite
        currentSpriteIndex = 0; // Reset indeks untuk animasi berikutnya
    }
    animationCounter++;
}

private void moveToPlayer() {
    if (target != null && !isDead) {
        int dx = target.getX() - getX(); // Jarak horizontal antara bos dan player
        int distance = Math.abs(dx); // Jarak absolut
        int speed = 3; // Kecepatan gerak bos

        // Logika menghadap ke arah player
        if (dx < 0 && isFacingRight) { // Player di kiri, bos harus menghadap kiri
            mirrorFacingDirection(false); // Ubah menghadap ke kiri
        } else if (dx > 0 && !isFacingRight) { // Player di kanan, bos harus menghadap kanan
            mirrorFacingDirection(true); // Ubah menghadap ke kanan
        }

        // Jika jarak cukup dekat, bos menyerang
        if (distance <= 90) {
            if (!attackOnCooldown) { // Hanya serang jika tidak dalam cooldown
                isAttacking = true;
                isMoving = false;
                attackPlayer(); // Panggil logika serangan
            }
        } else { // Jika tidak, bos bergerak mendekati player
            isAttacking = false;
            isMoving = true;

            // Update posisi bos
            setLocation(getX() + (dx > 0 ? speed : -speed), getY());
        }
    }

    // Reset cooldown jika sudah selesai
    if (attackOnCooldown && (System.currentTimeMillis() - lastAttackTime >= cooldownDuration)) {
        attackOnCooldown = false;
    }
}

private void mirrorFacingDirection(boolean faceRight) {
    if (faceRight != isFacingRight) { // Ubah arah hanya jika berbeda
        // Cerminkan sprite idle, gerakan, dan serangan
        idleSprite.mirrorHorizontally();
        moveSprites.mirrorHorizontally();
        for (GreenfootImage img : attackSprites) {
            img.mirrorHorizontally();
        }
        isFacingRight = faceRight; // Perbarui status arah
    }
}
```

```
    private void checkHitByAttack() {
        // Check if boss is hit by GatotAttackEffect
        GatotAttackEffect attack = (GatotAttackEffect) getOneIntersectingObject(GatotAttackEffect.class);
        if (attack != null) {
            health -= 1; // Decrease health
            BaseWorld world = (BaseWorld) getWorld();

            if (world != null) {
                world.removeObject(attack); // Remove the attack object
            }

            if (health <= 0) {
                world.getCounter().addScore(50);
                isDead = true; // Set boss as dead
                if (world != null) {
                    world.getCounter().addScore(10); // Bonus score for defeating boss
                    if (world instanceof BossFight4) {
                        ((BossFight4) world).stopMusic(); // Stop music
                    }
                }
                getWorld().removeObject(this); // Remove boss from the world
                Greenfoot.setWorld(new Level5());
            }
        }
    }

    private void stayOnGround() {
        if (getY() < groundY) {
            setLocation(getX(), groundY);
        }
    }
```

```
    private void attackPlayer() {
        if (isAttacking && !hasAttacked) { // Pastikan hanya menyerang satu kali per animasi
            Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
            if (player != null) {
                BaseWorld world = (BaseWorld) getWorld();
                if (world != null) {
                    HealthBar healthBar = world.getHealthBar();
                    healthBar.decreaseLife(); // Kurangi nyawa pemain hanya satu kali
                    hasAttacked = true; // Tandai bahwa boss telah menyerang
                }
            }
        }

        // Reset hasAttacked di akhir animasi serangan
        if (currentSpriteIndex == attackSprites.length - 1) {
            hasAttacked = false; // Siap untuk menyerang lagi di animasi berikutnya
        }
    }
}
```

## BossMonster5

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BossMonster5 extends Actor {
    private GreenfootImage idleSprite;
    private GreenfootImage[] moveSprites;
    private GreenfootImage[] attackSprites;
    private int currentSpriteIndex = 0;

    private int animationCounter = 1;
    private int animationDelay = 10;

    private boolean isAttacking = false;
    private boolean isMoving = false;
    private boolean isDead = false;
    public boolean isFacingRight = true; // Default boss menghadap ke kanan

    private int health = 1; // Health Boss
    private Gatot_StandBy target; // Reference to Gatot_StandBy
    private final int groundY = 350;

    private boolean attackOnCooldown = false; // Status cooldown
    private long lastAttackTime = 0; // Waktu serangan terakhir
    private final int cooldownDuration = 1000; // Cooldown dalam milidetik (2 detik)

    private boolean hasAttacked = false; // Status apakah sudah menyerang dalam satu siklus animasi
```

```
    public BossMonster5() {
        // Load idle sprite
        idleSprite = new GreenfootImage("boss5idle.png");
        idleSprite.scale(200, 200); // Perbesar sprite idle
        // Load movement sprites
        moveSprites = new GreenfootImage[7]; // Ukuran array harus sesuai dengan jumlah sprite gerakan
        for (int i = 0; i < 7; i++) { // Perbaiki jumlah sprite yang dimuat
            moveSprites[i] = new GreenfootImage("boss5move" + (i + 1) + ".png"); // Mulai dari 1 bukan 2
            moveSprites[i].scale(200, 200);
        }

        // Load attack sprites
        attackSprites = new GreenfootImage[4]; // Ukuran array sesuai jumlah sprite serangan
        for (int i = 0; i < 4; i++) { // Pastikan memuat 4 sprite
            attackSprites[i] = new GreenfootImage("boss5Attack" + (i + 1) + ".png");
            attackSprites[i].scale(200, 200);
        }
    }

    public void addedToWorld(World world) {
        // Get the Gatot_StandBy object
        target = (Gatot_StandBy) world.getObjects(Gatot_StandBy.class).get(0);
    }
```

```
public void act() {
    if (!isDead) {
        stayOnGround(); // Bos tetap berada di atas tanah
        moveToPlayer(); // Move towards Gatot_StandBy
        checkHitByAttack(); // Check if hit by GatotAttackEffect
    }

    // Handle animation
    if (isAttacking) {
        animate(attackSprites);
    } else if (isMoving) {
        animate(moveSprites);
    } else {
        animateIdle();
    }
}

private void animate(GreenfootImage[] sprites) {
    if (animationCounter % animationDelay == 0) {
        currentSpriteIndex = (currentSpriteIndex + 1) % sprites.length; // Hindari out of range
        setImage(sprites[currentSpriteIndex]);

        // Jika animasi selesai saat menyerang, aktifkan cooldown
        if (isAttacking && currentSpriteIndex == sprites.length - 1) {
            isAttacking = false;
            attackOnCooldown = true;
            lastAttackTime = System.currentTimeMillis();
            hasAttacked = false; // Reset status serangan untuk animasi berikutnya
        }
    }
    animationCounter++;
}
```

```
private void animateIdle() {
    if (animationCounter % animationDelay == 0) {
        setImage(idleSprite); // Idle hanya menggunakan satu sprite
        currentSpriteIndex = 0; // Reset indeks untuk animasi berikutnya
    }
    animationCounter++;
}

private void moveToPlayer() {
    if (target != null && !isDead) {
        int dx = target.getX() - getX(); // Jarak horizontal antara boss dan player
        int distance = Math.abs(dx); // Jarak absolut
        int speed = 1; // Kecepatan gerak boss

        // Periksa arah dan putar sesuai posisi target
        if (dx < 0 && !isFacingRight) { // Player di kiri, boss harus menghadap kiri
            mirrorFacingDirection(true); // Menghadap kiri
        } else if (dx > 0 && isFacingRight) { // Player di kanan, boss harus menghadap kanan
            mirrorFacingDirection(false); // Menghadap kanan
        }

        // Jika jarak cukup dekat, boss menyerang
        if (distance <= 90) {
            if (!attackOnCooldown) { // Hanya serang jika tidak dalam cooldown
                isAttacking = true;
                isMoving = false;
                attackPlayer(); // Panggil logika serangan
            }
        } else { // Jika tidak, boss bergerak mendekati player
            isAttacking = false;
            isMoving = true;
```

```
            // Update posisi boss
            setLocation(getX() + (dx > 0 ? speed : -speed), getY());
        }
    }

    // Reset cooldown jika sudah selesai
    if (attackOnCooldown && (System.currentTimeMillis() - lastAttackTime >= cooldownDuration)) {
        attackOnCooldown = false;
    }
}

private void mirrorFacingDirection(boolean faceRight) {
    if (faceRight != isFacingRight) { // Hanya ubah arah jika berbeda
        for (GreenfootImage img : moveSprites) {
            img.mirrorHorizontally(); // Balik semua sprite gerakan
        }
        idleSprite.mirrorHorizontally(); // Balik sprite idle
        for (GreenfootImage img : attackSprites) {
            img.mirrorHorizontally(); // Balik semua sprite serangan
        }
        isFacingRight = faceRight; // Perbarui status arah
    }
}
```

```
private void checkHitByAttack() {
    // Check if boss is hit by GatotAttackEffect
    GatotAttackEffect attack = (GatotAttackEffect) getOneIntersectingObject(GatotAttackEffect.class);
    if (attack != null) {
        health -= 1; // Decrease health
        BaseWorld world = (BaseWorld) getWorld();

        if (world != null) {
            world.removeObject(attack); // Remove the attack object
        }

        if (health <= 0) {
            world.getCounter().addScore(50);
            isDead = true; // Set boss as dead
            if (world != null) {
                world.getCounter().addScore(10); // Bonus score for defeating boss
                if (world instanceof BossFight5) {
                    ((BossFight5) world).stopMusic(); // Stop music
                }
            }
            getWorld().removeObject(this); // Remove boss from the world

            int finalScore = world.getCounter().getScore();
            Greenfoot.setWorld(new victorious(finalScore));
        }
    }
}
```

```
    private void stayOnGround() {
        if (getY() < groundY) {
            setLocation(getX(), groundY);
        }
    }

    private void attackPlayer() {
        if (isAttacking && !hasAttacked) { // Pastikan hanya menyerang satu kali per animasi
            Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
            if (player != null) {
                BaseWorld world = (BaseWorld) getWorld();
                if (world != null) {
                    HealthBar healthBar = world.getHealthBar();
                    healthBar.decreaseLife(); // Kurangi nyawa pemain hanya satu kali
                    hasAttacked = true; // Tandai bahwa boss telah menyerang
                }
            }
        }

        // Reset hasAttacked di akhir animasi serangan
        if (currentSpriteIndex == attackSprites.length - 1) {
            hasAttacked = false; // Siap untuk menyerang lagi di animasi berikutnya
        }
    }
}
```

**Coin (Objek yang diletakkan pada beberapa world bertujuan untuk menambah skor pemain)**

Coin

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3  public class Coin extends Actor {
4      public void act() {
5          checkTouchByGatot();
6      }
7
8      private void checkTouchByGatot() {
9          // yang bisa ambil gatotstandby & gatotlabyrntine
10         Gatot_StandBy gatotStandBy = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
11         Gatot_Labyrntine gatotLabyrntine = (Gatot_Labyrntine) getOneIntersectingObject(Gatot_Labyrntine.class);
12
13         if (gatotStandBy != null) {
14             // Akses dunia dan counter
15             BaseWorld world = (BaseWorld) getWorld();
16             Counter counter = world.getCounter();
17
18             if (counter != null) {
19                 counter.addScore(3); // 3 skor untuk Gatot_StandBy
20             }
21
22             world.removeObject(this); // Hapus objek koin klo disentuh
23         } else if (gatotLabyrntine != null) {
24             // Akses dunia dan counter
25             BaseWorld world = (BaseWorld) getWorld();
26             Counter counter = world.getCounter();
27
28             if (counter != null) {
29                 counter.addScore(2); // 2 skor untuk Gatot_Labyrintine
30             }
31
32             world.removeObject(this); // Hapus objek koin klo disentuh
33         }
34     }
35 }
```

Penjelasan :

```
public void act() {
    checkTouchByGatot();
```

```
    }
```
> Dipanggil checkTouchByGatot() untuk mengecek apakah koin disentuh oleh Gatot_StandBy atau Gatot_Labyrntine.

```
private void checkTouchByGatot() {
      // yang bisa ambil gatotstandby & gatotlabyrntine
                                        Gatot_StandBy gatotStandBy = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);
                                    Gatot_Labyrntine gatotLabyrntine = (Gatot_Labyrntine) getOneIntersectingObject(Gatot_Labyrntine.class);
```
> checkTouchByGatot() memeriksa apakah objek Coin bersentuhan dengan Gatot_StandBy atau Gatot_Labyrntine.
> getOneIntersectingObject(Gatot_StandBy.class) mengembalikan objek Gatot_StandBy jika koin menyentuh karakter tersebut. Demikian pula untuk Gatot_Labyrntine.

```
if (gatotStandBy != null) {
        // Akses dunia dan counter
        BaseWorld world = (BaseWorld) getWorld();
        Counter counter = world.getCounter();

        if (counter != null) {
           counter.addScore(3); // 3 skor untuk Gatot_StandBy
        }

        world.removeObject(this); // Hapus objek koin klo disentuh
```
> Jika gatotStandBy bukan null, artinya Gatot_StandBy menyentuh koin.
> Program mengakses BaseWorld, untuk mendapatkan objek Counter agar bisa mengatur skor.
> counter.addScore(3) menambah skor sebanyak 3.
> Koin kemudian dihapus dari dunia dengan world.removeObject(this).

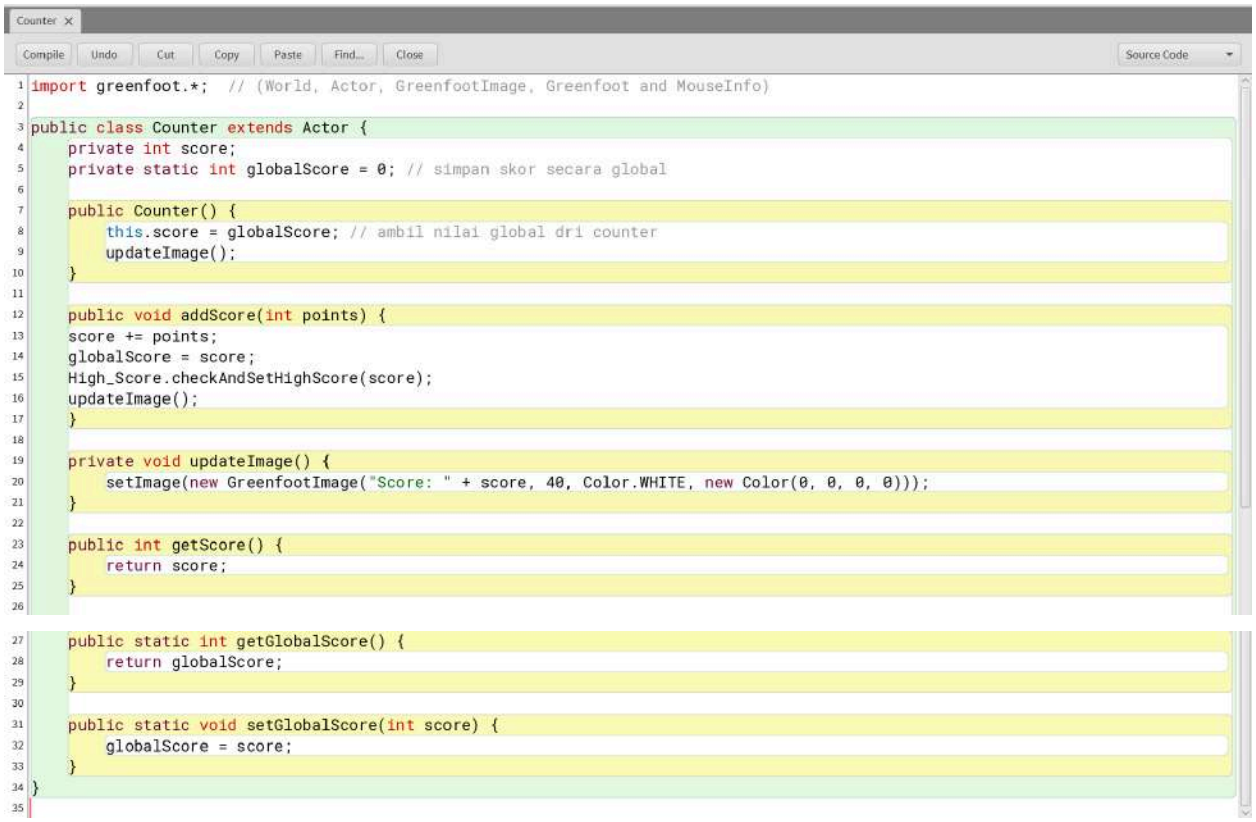
```
} else if (gatotLabyrntine != null) {
        // Akses dunia dan counter
        BaseWorld world = (BaseWorld) getWorld();
        Counter counter = world.getCounter();

        if (counter != null) {
           counter.addScore(2); // 2 skor untuk Gatot_Labyrintine
        }

        world.removeObject(this); // Hapus objek koin klo disentuh
```

```
    }
```

> Jika gatotLabyrntine bukan null, artinya Gatot_Labyrntine menyentuh koin.

> Logika yang sama dengan saat Gatot_StandBy yang terjadi: mengakses dunia dan Counter, menambah skor sebanyak 2, lalu menghapus koin dari dunia.

**Counter (Menyimpan skor)**



```
Counter ×
Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3  public class Counter extends Actor {
4      private int score;
5      private static int globalScore = 0; // simpan skor secara global
6
7      public Counter() {
8          this.score = globalScore; // ambil nilai global dri counter
9          updateImage();
10     }
11
12     public void addScore(int points) {
13     score += points;
14     globalScore = score;
15     High_Score.checkAndSetHighScore(score);
16     updateImage();
17     }
18
19     private void updateImage() {
20         setImage(new GreenfootImage("Score: " + score, 40, Color.WHITE, new Color(0, 0, 0, 0)));
21     }
22
23     public int getScore() {
24         return score;
25     }
26
27     public static int getGlobalScore() {
28         return globalScore;
29     }
30
31     public static void setGlobalScore(int score) {
32         globalScore = score;
33     }
34 }
35
```

Penjelasan :

```
private int score;
private static int globalScore = 0; // simpan skor secara global
```

> score: Menyimpan skor lokal untuk objek Counter ini.

> globalScore: Variabel statis yang menyimpan skor global yang digunakan untuk melacak skor di world level yang membutuhkan skor. Variabel ini bersifat statis, artinya nilainya dapat diakses tanpa memerlukan instance objek.

```
public Counter() {
    this.score = globalScore; // ambil nilai global dri counter
    updateImage();
  }
```

> Counter dipanggil saat objek Counter dibuat. Konstruktor ini menginisialisasi score dengan nilai globalScore yang disimpan secara global, sehingga Counter ini akan menampilkan skor

yang sama dengan nilai global yang ada saat itu. Setelah skor diatur, updateImage() dipanggil untuk memperbarui tampilan objek (gambar teks skor).

```
public void addScore(int points) {
   score += points;
   globalScore = score;
   High_Score.checkAndSetHighScore(score);
   updateImage();
   }
```

> addScore(int points) digunakan untuk menambah skor score lokal dan globalScore global.
> score += points menambahkan skor lokal dengan nilai yang diberikan pada parameter points.
> globalScore = score memperbarui skor global setelah menambahkannya.
> High_Score.checkAndSetHighScore(score) memanggil metode untuk memeriksa apakah skor saat ini lebih tinggi dari skor tertinggi sebelumnya dan memperbarui skor tertinggi jika perlu.
> updateImage() dipanggil untuk memperbarui gambar objek dan menampilkan skor yang baru.

```
private void updateImage() {
        setImage(new GreenfootImage("Score: " + score, 40, Color.WHITE, new Color(0, 0, 0, 0)));
   }
```

> Menggunakan GreenfootImage, objek gambar baru dibuat dengan teks "Score: [score]" di mana score adalah nilai skor yang ada saat itu. Gambar ini menggunakan ukuran font 40, warna teks putih Color.WHITE, dan latar belakang transparan dengan new Color(0, 0, 0, 0).

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

> getScore() mengembalikan nilai score lokal dari objek Counter ini, yang bisa dipanggil untuk mengetahui skor yang disimpan dalam objek tersebut.

```
   public static int getGlobalScore() {
      return globalScore;
   }
```

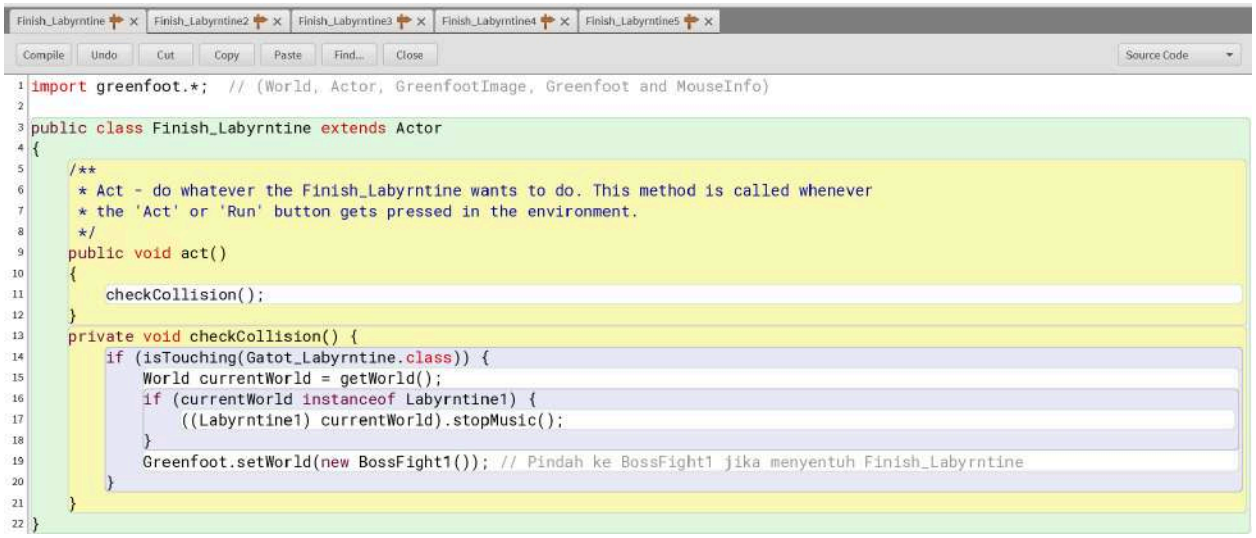
> getGlobalScore() adalah metode statis yang mengembalikan nilai globalScore. Karena globalScore bersifat statis, metode ini dapat diakses tanpa harus membuat instance Counter.

```
public static void setGlobalScore(int score) {
      globalScore = score;
   }
```

> setGlobalScore(int score) adalah metode statis yang memungkinkan untuk mengatur nilai globalScore secara manual dari luar kelas Counter. Karena bersifat statis, ini mengubah skor global yang digunakan oleh semua instance objek Counter.

**Finish_Labyrntine (Objek yang berada pada world "Labyrntine")**
**Terdapat 5 objek ini pada Actor dengan kode Java yang sama, perbedaannya hanya transisi worldnya yang menyesuaikan level "Labyrntine"**

- Finish_Labyrntine berada pada Labyrntine1 dengan transisi ke BossFight1
- Finish_Labyrntine2 berada pada Labyrntine2 dengan transisi ke BossFight2
- Finish_Labyrntine3 berada pada Labyrntine3 dengan transisi ke BossFight3
- Finish_Labyrntine4 berada pada Labyrntine4 dengan transisi ke BossFight4
- Finish_Labyrntine5 berada pada Labyrntine5 dengan transisi ke BossFight5




Penjelasan :

```
public void act()
   {
      checkCollision();
   }
```

> Memanggil checkCollision.

```
private void checkCollision() {
      if (isTouching(Gatot_Labyrntine.class)) {
         World currentWorld = getWorld();
         if (currentWorld instanceof Labyrntine1) {
            ((Labyrntine1) currentWorld).stopMusic();
         }
               Greenfoot.setWorld(new BossFight1()); // Pindah ke BossFight1 jika menyentuh
Finish_Labyrntine
```

```
    }
}
```

> Memeriksa apakah objek Finish_Labyrntine menyentuh objek Gatot_Labyrntine. Jika terjadi sentuhan, pertama-tama, dunia saat ini (currentWorld) diambil dengan getWorld(). Jika dunia tersebut adalah instance dari Labyrntine1, maka metode stopMusic() dipanggil untuk menghentikan musik yang sedang diputar. Setelah itu, permainan akan beralih ke dunia baru, BossFight1, dengan menggunakan Greenfoot.setWorld(new BossFight1()).

**Finish_Labyrinthine2**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Finish_Labyrntine here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Finish_Labyrntine2 extends Actor
{
    /**
     * Act - do whatever the Finish_Labyrntine wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkCollision();
    }
    private void checkCollision() {
        if (isTouching(Gatot_Labyrntine.class)) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Labyrntine2) {
                ((Labyrntine2) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new BossFight2()); // Pindah ke BossFight2 jika menyentuh Finish_Labyrntine
        }
    }
}
```

## Finish_Labyrinthine3

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Finish_Labyrntine here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Finish_Labyrntine3 extends Actor
{
    /**
     * Act - do whatever the Finish_Labyrntine wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkCollision();
    }
    private void checkCollision() {
        if (isTouching(Gatot_Labyrntine.class)) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Labyrntine3) {
                ((Labyrntine3) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new BossFight3()); // Pindah ke BossFight3 jika menyentuh Finish_Labyrntine
        }
    }
}
```

## Finish_Labyrinhine4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Finish_Labyrntine here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Finish_Labyrntine4 extends Actor
{
    /**
     * Act - do whatever the Finish_Labyrntine wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkCollision();
    }
    private void checkCollision() {
        if (isTouching(Gatot_Labyrntine.class)) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Labyrntine4) {
                ((Labyrntine4) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new BossFight4()); // Pindah ke BossFight4 jika menyentuh Finish_Labyrntine
        }
    }
}
```

**Finish_Labyrinthine5**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Finish_Labyrntine here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Finish_Labyrntine5 extends Actor
{
    /**
     * Act - do whatever the Finish_Labyrntine wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkCollision();
    }
    private void checkCollision() {
        if (isTouching(Gatot_Labyrntine.class)) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Labyrntine5) {
                ((Labyrntine5) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new BossFight5()); // Pindah ke BossFight5 jika menyentuh Finish_Labyrntine
        }
    }
}
```

**GameOver**

GameOver

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
1  import greenfoot.*;
2  import java.awt.Color;
3
4  public class GameOver extends Actor {
5      private String playerName = "";
6      private GreenfootImage displayImage;
7      private int playerScore;
8
9      public GameOver(int score) {
10         this.playerScore = score;
11         updateMessage();
12     }
13
14     public void act() {
15         String key = Greenfoot.getKey();
16         if (key != null) {
17             if (key.equals("enter") && !playerName.isEmpty()) {
18                 historyMenu.addHighScore(playerName, playerScore);
19                 Greenfoot.setWorld(new historyMenu());
20             } else if (key.equals("backspace") && playerName.length() > 0) {
21                 playerName = playerName.substring(0, playerName.length() - 1);
22             } else if (key.length() == 1 && playerName.length() < 10) { // Maksimal 10 karakter
23                 playerName += key.toUpperCase();
24             }
25             updateMessage();
26         }
27     }
```

Penjelasan :

private String playerName = "";

private GreenfootImage displayImage;

private int playerScore;

> playerName: Menyimpan nama pemain yang dimasukkan selama world Game_Over.

> displayImage: Gambar yang akan ditampilkan pada world Game_Over.

> playerScore: Menyimpan skor pemain yang dicapai selama permainan.

```
public GameOver(int score) {
        this.playerScore = score;
        updateMessage();
    }
```
> Menerima skor pemain sebagai parameter dan menyimpannya di playerScore. Setelah itu, memanggil updateMessage() untuk memperbarui tampilan Game Over dengan informasi skor dan tempat untuk memasukkan nama pemain.

```
public void act() {
        String key = Greenfoot.getKey();
        if (key != null) {
            if (key.equals("enter") && !playerName.isEmpty()) {
                historyMenu.addHighScore(playerName, playerScore);
                Greenfoot.setWorld(new historyMenu());
            } else if (key.equals("backspace") && playerName.length() > 0) {
                playerName = playerName.substring(0, playerName.length() - 1);
            } else if (key.length() == 1 && playerName.length() < 10) { // Maksimal 10 karakter
                playerName += key.toUpperCase();
            }
            updateMessage();
        }
    }
```
> Memeriksa input dari pemain dengan Greenfoot.getKey().
> Jika tombol Enter ditekan dan nama pemain sudah dimasukkan, skor akan disimpan melalui historyMenu.addHighScore(), dan permainan beralih ke dunia historyMenu.
> Jika tombol Backspace ditekan, nama pemain akan dihapus karakter terakhirnya.
> Jika tombol lain yang merupakan karakter tunggal ditekan dan panjang nama pemain kurang dari 10 karakter, maka karakter tersebut ditambahkan ke nama pemain.
> Setelah itu, updateMessage() dipanggil lagi untuk memperbarui tampilan Game Over.

```
28
29      private void updateMessage() {
30          displayImage = new GreenfootImage(600, 200); // Set width dan height
31          displayImage.setColor(greenfoot.Color.BLACK);  // Gunakan warna solid
32          displayImage.fill();                           // Fill background
33          displayImage.setColor(greenfoot.Color.RED);    // Judul merah
34          displayImage.setFont(displayImage.getFont().deriveFont(50f));
35          displayImage.drawString("GAME OVER", 150, 50);
36
37          displayImage.setColor(greenfoot.Color.WHITE);
38          displayImage.setFont(displayImage.getFont().deriveFont(30f));
39          displayImage.drawString("Enter Name: " + playerName, 100, 120);
40          displayImage.drawString("Score: " + playerScore, 100, 170);
41
42          setImage(displayImage);
43      }
44  }
```

Penjelasan :

```
private void updateMessage() {
        displayImage = new GreenfootImage(600, 200); // Set width dan height
        displayImage.setColor(greenfoot.Color.BLACK);  // Gunakan warna solid
        displayImage.fill();                           // Fill background
        displayImage.setColor(greenfoot.Color.RED);    // Judul merah
        displayImage.setFont(displayImage.getFont().deriveFont(50f));
        displayImage.drawString("GAME OVER", 150, 50);

        displayImage.setColor(greenfoot.Color.WHITE);
        displayImage.setFont(displayImage.getFont().deriveFont(30f));
        displayImage.drawString("Enter Name: " + playerName, 100, 120);
        displayImage.drawString("Score: " + playerScore, 100, 170);

        setImage(displayImage);
    }
```

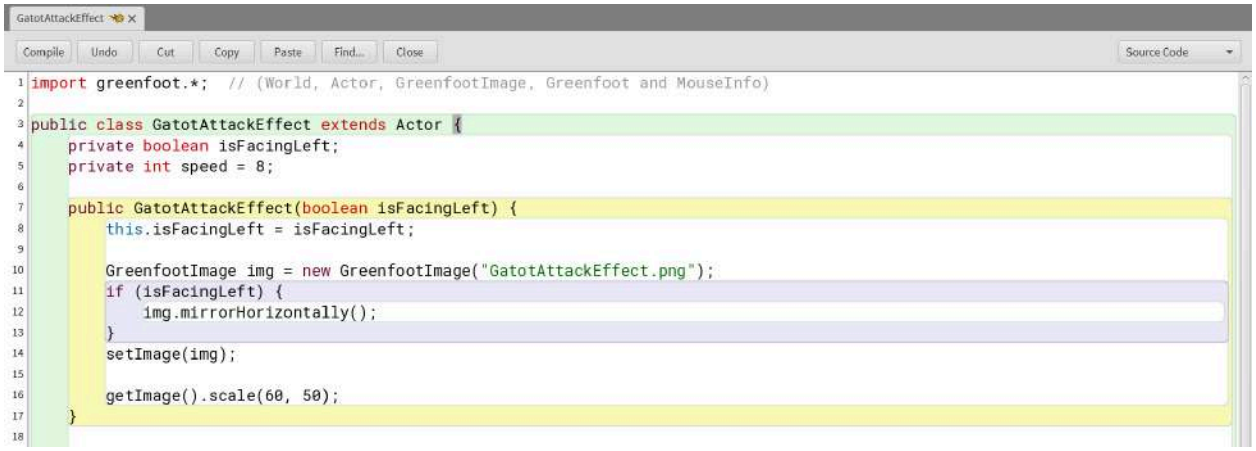
> Membuat displayImage dengan ukuran 600x200 piksel, yang akan menampilkan pesan "GAME OVER", nama pemain yang sedang dimasukkan, dan skor pemain.

> Latar belakang gambar diisi dengan warna hitam, lalu teks judul "GAME OVER" ditampilkan dengan warna merah, menggunakan font besar.

> Nama pemain dan skor ditampilkan dalam warna putih dengan font yang lebih kecil.

> Gambar displayImage kemudian diterapkan pada objek ini dengan setImage(), yang akan menampilkan gambar.

**GatotAttackEffect (Serangan player untuk membunuh / menghilangkan musuh dari world)**




```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class GatotAttackEffect extends Actor {
    private boolean isFacingLeft;
    private int speed = 8;

    public GatotAttackEffect(boolean isFacingLeft) {
        this.isFacingLeft = isFacingLeft;

        GreenfootImage img = new GreenfootImage("GatotAttackEffect.png");
        if (isFacingLeft) {
            img.mirrorHorizontally();
        }
        setImage(img);

        getImage().scale(60, 50);
    }
```

Penjelasan :

```
private boolean isFacingLeft;
private int speed = 8;
```

> isFacingLeft: Boolean yang menyimpan apakah objek serangan sedang menghadap kiri (true) atau kanan (false).

> speed: Kecepatan pergerakan objek serangan. Nilainya adalah 8.

```
public GatotAttackEffect(boolean isFacingLeft) {
      this.isFacingLeft = isFacingLeft;

      GreenfootImage img = new GreenfootImage("GatotAttackEffect.png");
      if (isFacingLeft) {
         img.mirrorHorizontally();
      }
      setImage(img);

      getImage().scale(60, 50);
   }
```

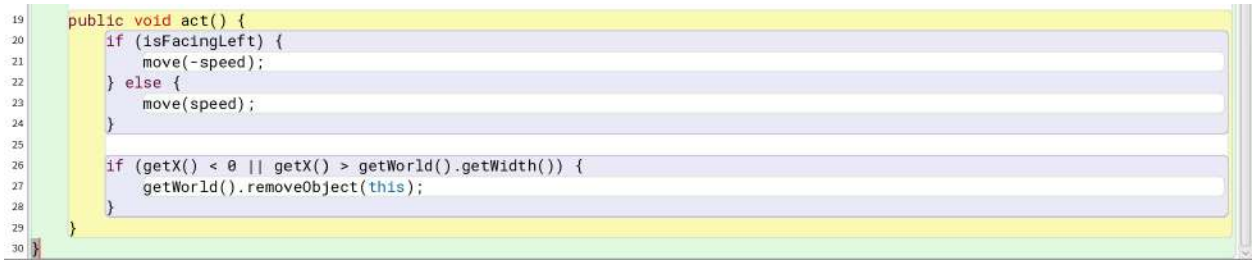
> Menerima parameter isFacingLeft yang menentukan apakah objek serangan ini menghadap kiri atau kanan.

> GreenfootImage img = new GreenfootImage("GatotAttackEffect.png");: Membuat objek gambar dari file "GatotAttackEffect.png", yang akan digunakan sebagai sprite serangan.

> Jika isFacingLeft bernilai true, gambar akan diputar secara horizontal menggunakan img.mirrorHorizontally(); untuk membalikkan gambar.

> setImage(img);: Menetapkan gambar yang sudah diproses ke objek ini.

> getImage().scale(60, 50);: Mengubah ukuran gambar menjadi 60x50 piksel agar sesuai dengan ukuran yang diinginkan untuk efek serangan.



```
    public void act() {
        if (isFacingLeft) {
            move(-speed);
        } else {
            move(speed);
        }

        if (getX() < 0 || getX() > getWorld().getWidth()) {
            getWorld().removeObject(this);
        }
    }
}
```

Penjelasan :

```
public void act() {
      if (isFacingLeft) {
         move(-speed);
      } else {
         move(speed);
      }

      if (getX() < 0 || getX() > getWorld().getWidth()) {
         getWorld().removeObject(this);
      }
   }
```

> if (isFacingLeft): Jika objek serangan menghadap kiri, maka objek akan bergerak ke kiri dengan kecepatan yang didefinisikan oleh variabel speed (menggunakan move(-speed);).
> Jika objek menghadap kanan (yaitu, isFacingLeft bernilai false), objek akan bergerak ke kanan dengan kecepatan yang sama (menggunakan move(speed);).
> if (getX() < 0 || getX() > getWorld().getWidth()): Jika objek bergerak keluar dari layar (lewat batas kiri atau kanan dunia), objek serangan akan dihapus dari dunia menggunakan getWorld().removeObject(this);.

**Gatot_Labyrntine (Gatot yang dikontrol oleh player pada world “Labyrntine”)**

Gatot_Labyrntine

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot, and MouseInfo)

public class Gatot_Labyrntine extends Actor {
    private int speed = 5; // Kecepatan gerakan
    private int moveDirectionX = 0; // Arah gerakan x
    private int moveDirectionY = 0; // Arah gerakan y

    public Gatot_Labyrntine(int height, int width) {
        getImage().scale(height, width);
    }

    public void act() {
        handleInput();
        moveStep();
    }
```

Penjelasan :

```
private int speed = 5; // Kecepatan gerakan
private int moveDirectionX = 0; // Arah gerakan x
```

```
private int moveDirectionY = 0; // Arah gerakan y
```
> speed: Menyimpan kecepatan gerakan Gatot_Labyrntine. Nilainya adalah 5.
> moveDirectionX: Menyimpan arah gerakan pada sumbu X (horizontal). Nilainya diatur berdasarkan input pemain.
moveDirectionY: Menyimpan arah gerakan pada sumbu Y (vertikal). Nilainya juga diatur berdasarkan input pemain.

```
public Gatot_Labyrntine(int height, int width) {
        getImage().scale(height, width);
    }
```
> Mengatur ukuran objek yang diletakkan pada world.

```
public void act() {
        handleInput();
        moveStep();
    }
```
> Memanggil handleInput(); dan moveStep();  .

```
17      private void handleInput() {
18          if (moveDirectionX == 0 && moveDirectionY == 0) {
19              if (Greenfoot.isKeyDown("up")) {
20                  moveDirectionX = 0;
21                  moveDirectionY = -speed;
23              } else if (Greenfoot.isKeyDown("down")) {
24                  moveDirectionX = 0;
25                  moveDirectionY = speed;
26              } else if (Greenfoot.isKeyDown("left")) {
27                  moveDirectionX = -speed;
28                  moveDirectionY = 0;
29              } else if (Greenfoot.isKeyDown("right")) {
30                  moveDirectionX = speed;
31                  moveDirectionY = 0;
32              }
33          }
34      }
```

Penjelasan :

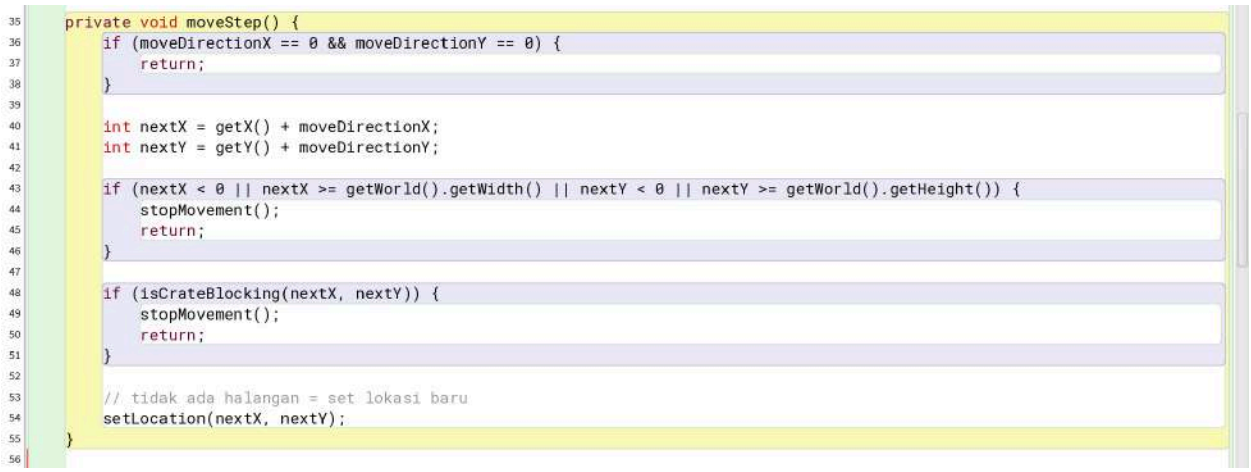
```
private void handleInput() {
        if (moveDirectionX == 0 && moveDirectionY == 0) {
            if (Greenfoot.isKeyDown("up")) {
                moveDirectionX = 0;
                moveDirectionY = -speed;
            } else if (Greenfoot.isKeyDown("down")) {
                moveDirectionX = 0;
                moveDirectionY = speed;
            } else if (Greenfoot.isKeyDown("left")) {
                moveDirectionX = -speed;
                moveDirectionY = 0;
            } else if (Greenfoot.isKeyDown("right")) {
```

```
            moveDirectionX = speed;
            moveDirectionY = 0;
        }
    }
}
```

> Memeriksa apakah Gatot sedang tidak bergerak (moveDirectionX == 0 && moveDirectionY == 0), dan jika tidak bergerak, metode ini akan memeriksa apakah tombol arah (up, down, left, right) sedang ditekan.

> Jika tombol arah ditekan, nilai moveDirectionX dan moveDirectionY akan diubah sesuai dengan tombol yang ditekan.

- Tombol "up": Gerakan ke atas (vertikal) dengan moveDirectionY = -speed.
- Tombol "down": Gerakan ke bawah (vertikal) dengan moveDirectionY = speed.
- Tombol "left": Gerakan ke kiri (horizontal) dengan moveDirectionX = -speed.
- Tombol "right": Gerakan ke kanan (horizontal) dengan moveDirectionX = speed.



```
private void moveStep() {
    if (moveDirectionX == 0 && moveDirectionY == 0) {
        return;
    }

    int nextX = getX() + moveDirectionX;
    int nextY = getY() + moveDirectionY;

    if (nextX < 0 || nextX >= getWorld().getWidth() || nextY < 0 || nextY >= getWorld().getHeight()) {
        stopMovement();
        return;
    }

    if (isCrateBlocking(nextX, nextY)) {
        stopMovement();
        return;
    }

    // tidak ada halangan = set lokasi baru
    setLocation(nextX, nextY);
}
```

Penjelasan :

```
private void moveStep() {
    if (moveDirectionX == 0 && moveDirectionY == 0) {
        return;
    }

    int nextX = getX() + moveDirectionX;
    int nextY = getY() + moveDirectionY;

        if (nextX < 0 || nextX >= getWorld().getWidth() || nextY < 0 || nextY >= getWorld().getHeight()) {
        stopMovement();
        return;
    }
```

```
        if (isCrateBlocking(nextX, nextY)) {
            stopMovement();
            return;
        }

        // tidak ada halangan = set lokasi baru
        setLocation(nextX, nextY);
    }
```

> Memeriksa apakah Gatot memiliki gerakan yang sedang aktif (apakah moveDirectionX atau moveDirectionY bukan 0).
> Menghitung posisi baru karakter berdasarkan arah gerakan (nextX dan nextY).
> Memeriksa apakah posisi baru keluar dari batas dunia menggunakan getWorld().getWidth() dan getWorld().getHeight(). Jika keluar, gerakan dihentikan.
> Memeriksa apakah ada objek Crate yang menghalangi gerakan karakter dengan memanggil metode isCrateBlocking(). Jika ada halangan, gerakan dihentikan.
> Jika tidak ada halangan, karakter dipindahkan ke posisi baru dengan setLocation(nextX, nextY).

```
    private boolean isCrateBlocking(int nextX, int nextY) {
        int gatotLeft = nextX - getImage().getWidth() / 2;
        int gatotRight = nextX + getImage().getWidth() / 2;
        int gatotTop = nextY - getImage().getHeight() / 2;
        int gatotBottom = nextY + getImage().getHeight() / 2;

        for (Object obj : getWorld().getObjects(Crate.class)) {
            Crate crate = (Crate) obj;
            int crateLeft = crate.getX() - crate.getImage().getWidth() / 2;
            int crateRight = crate.getX() + crate.getImage().getWidth() / 2;
            int crateTop = crate.getY() - crate.getImage().getHeight() / 2;
            int crateBottom = crate.getY() + crate.getImage().getHeight() / 2;

            // Periksa sisi sesuai arah gerakan
            if (moveDirectionX < 0 && gatotLeft <= crateRight && gatotRight > crateLeft && gatotBottom > crateTop && gatotTop < c
                return true; // Gerakan ke kiri
            }
            if (moveDirectionX > 0 && gatotRight >= crateLeft && gatotLeft < crateRight && gatotBottom > crateTop && gatotTop < c
                return true; // Gerakan ke kanan
            }
            if (moveDirectionY < 0 && gatotTop <= crateBottom && gatotBottom > crateTop && gatotLeft < crateRight && gatotRight >
                return true; // Gerakan ke atas
            }
            if (moveDirectionY > 0 && gatotBottom >= crateTop && gatotTop < crateBottom && gatotLeft < crateRight && gatotRight >
                return true; // Gerakan ke bawah
            }
        }
        return false; // Tidak ada crate yang menghalangi
    }

    private void stopMovement() {
        moveDirectionX = 0;
        moveDirectionY = 0;
    }
}
```

Penjelasan :
private boolean isCrateBlocking(int nextX, int nextY) {

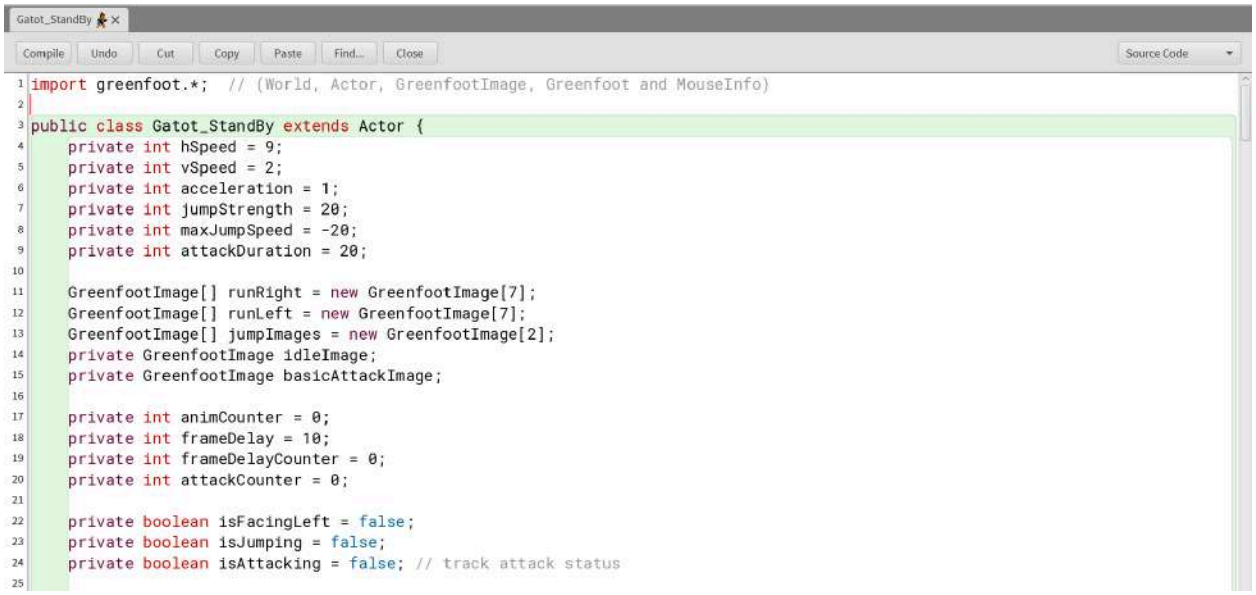
```
        int gatotLeft = nextX - getImage().getWidth() / 2;
        int gatotRight = nextX + getImage().getWidth() / 2;
        int gatotTop = nextY - getImage().getHeight() / 2;
        int gatotBottom = nextY + getImage().getHeight() / 2;

        for (Object obj : getWorld().getObjects(Crate.class)) {
            Crate crate = (Crate) obj;
            int crateLeft = crate.getX() - crate.getImage().getWidth() / 2;
            int crateRight = crate.getX() + crate.getImage().getWidth() / 2;
            int crateTop = crate.getY() - crate.getImage().getHeight() / 2;
            int crateBottom = crate.getY() + crate.getImage().getHeight() / 2;

            // Periksa sisi sesuai arah gerakan
                if (moveDirectionX < 0 && gatotLeft <= crateRight && gatotRight > crateLeft &&
gatotBottom > crateTop && gatotTop < crateBottom) {
                return true; // Gerakan ke kiri
            }
                if (moveDirectionX > 0 && gatotRight >= crateLeft && gatotLeft < crateRight &&
gatotBottom > crateTop && gatotTop < crateBottom) {
                return true; // Gerakan ke kanan
            }
              if (moveDirectionY < 0 && gatotTop <= crateBottom && gatotBottom > crateTop &&
gatotLeft < crateRight && gatotRight > crateLeft) {
                return true; // Gerakan ke atas
            }
              if (moveDirectionY > 0 && gatotBottom >= crateTop && gatotTop < crateBottom &&
gatotLeft < crateRight && gatotRight > crateLeft) {
                return true; // Gerakan ke bawah
            }
        }
        return false; // Tidak ada crate yang menghalangi
    }
```

> Memeriksa apakah objek Gatot_Labyrntine bertabrakan dengan objek Crate saat bergerak. Fungsi isCrateBlocking menghitung batas-batas (kiri, kanan, atas, bawah) dari Gatot_Labyrntine dan setiap Crate di dunia, kemudian memeriksa apakah ada tumpang tindih antara keduanya berdasarkan arah gerakan. Jika terdapat tabrakan (misalnya saat bergerak ke kiri, kanan, atas, atau bawah), fungsi ini akan mengembalikan true, yang menandakan bahwa pergerakan harus dihentikan. Fungsi stopMovement digunakan untuk menghentikan pergerakan dengan mengatur moveDirectionX dan moveDirectionY ke 0, sehingga objek berhenti bergerak jika ada halangan.

```
private void stopMovement() {
        moveDirectionX = 0;
        moveDirectionY = 0;
    }
```

> Mengatur moveDirectionX dan moveDirectionY ke 0, yang menghentikan pergerakan pada frame berikutnya.

**Gatot_StandBy (Karakter pemain yang berada pada world “Level” dan juga “BossFight”)**




```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Gatot_StandBy extends Actor {
    private int hSpeed = 9;
    private int vSpeed = 2;
    private int acceleration = 1;
    private int jumpStrength = 20;
    private int maxJumpSpeed = -20;
    private int attackDuration = 20;

    GreenfootImage[] runRight = new GreenfootImage[7];
    GreenfootImage[] runLeft = new GreenfootImage[7];
    GreenfootImage[] jumpImages = new GreenfootImage[2];
    private GreenfootImage idleImage;
    private GreenfootImage basicAttackImage;

    private int animCounter = 0;
    private int frameDelay = 10;
    private int frameDelayCounter = 0;
    private int attackCounter = 0;

    private boolean isFacingLeft = false;
    private boolean isJumping = false;
    private boolean isAttacking = false; // track attack status
```

Penjelasan :

```
    private int hSpeed = 9;
    private int vSpeed = 2;
    private int acceleration = 1;
    private int jumpStrength = 20;
    private int maxJumpSpeed = -20;
    private int attackDuration = 20;

    GreenfootImage[] runRight = new GreenfootImage[7];
    GreenfootImage[] runLeft = new GreenfootImage[7];
    GreenfootImage[] jumpImages = new GreenfootImage[2];
    private GreenfootImage idleImage;
    private GreenfootImage basicAttackImage;

    private int animCounter = 0;
    private int frameDelay = 10;
    private int frameDelayCounter = 0;
    private int attackCounter = 0;
```

```
    private boolean isFacingLeft = false;
    private boolean isJumping = false;
    private boolean isAttacking = false; // track attack status
```

> hSpeed dan vSpeed: Kecepatan horizontal dan vertikal pergerakan karakter.
> acceleration: Akselerasi jatuh untuk simulasi gravitasi.
> jumpStrength dan maxJumpSpeed: Mengatur kekuatan lompatan dan kecepatan maksimum jatuh.
> attackDuration: Durasi serangan.
> runRight[] dan runLeft[]: Array untuk menyimpan sprite animasi karakter yang berlari ke kanan dan kiri.
> jumpImages[]: Array untuk menyimpan sprite animasi lompatan.
> idleImage dan basicAttackImage: Gambar untuk status diam dan serangan dasar.
> animCounter: Counter untuk animasi, mengontrol urutan frame.
> frameDelay dan frameDelayCounter: Mengontrol kecepatan perubahan frame animasi.
> attackCounter: Menyimpan waktu durasi serangan.
> isFacingLeft, isJumping, isAttacking: Variabel boolean untuk melacak kondisi karakter (apakah menghadap kiri, sedang melompat, atau sedang menyerang).

```
26      public Gatot_StandBy() {
27          initAnimationSprites();
28          idleImage = new GreenfootImage("Gatot_StandBy.png");
29          basicAttackImage = new GreenfootImage("GatotBasicAttack.png");
30          basicAttackImage.scale(100, 130);
31          setImage(idleImage);
32      }
33
34      public Gatot_StandBy(int height, int width) {
35          this();
36          idleImage.scale(height, width);
37      }
38
39      public void act() {
40          if (isAttacking) {
41              attackCounter--;
42              if (attackCounter <= 0) {
43                  isAttacking = false;
44                  setIdleImage();
45              }
46          } else {
47              checkKeys();
48          }
49          checkFall();
50      }
```

Penjelasan :

```
public Gatot_StandBy(int height, int width) {
        this();
        idleImage.scale(height, width);
    }
```

> Scale ukuran Gatot_StandBy.

```
public void act() {
        if (isAttacking) {
```

```
            attackCounter--;
            if (attackCounter <= 0) {
                isAttacking = false;
                setIdleImage();
            }
        } else {
            checkKeys();
        }
        checkFall();
    }
```

> isAttacking: Jika karakter sedang menyerang, durasi serangan akan berkurang hingga selesai, dan karakter akan kembali ke posisi idle.

> checkKeys(): Memeriksa apakah tombol arah atau tombol lainnya (seperti "space" untuk melompat atau mouse untuk menyerang) ditekan, dan menggerakkan karakter sesuai input tersebut.

> checkFall(): Memeriksa apakah karakter menyentuh tanah dan menyesuaikan posisi vertikalnya agar tidak terus jatuh jika sudah di tanah.

```
52      public void checkKeys() {
53          boolean isMoving = false;
54
55          if (Greenfoot.isKeyDown("d")) {
56              moveRight();
57              animateRunRight();
58              isFacingLeft = false;
59              isMoving = true;
60          }
61          if (Greenfoot.isKeyDown("a")) {
62              moveLeft();
63              animateRunLeft();
64              isFacingLeft = true;
65              isMoving = true;
66          }
67          if (Greenfoot.isKeyDown("space")) {
68              jump();
69          }
70          if (Greenfoot.mousePressed(null)) {
71              performAttack();
72          }
73
74          if (!isMoving && !isJumping && !isAttacking) {
75              setIdleImage();
76          }
77      }
```

Penjelasan :

```
public void checkKeys() {
        boolean isMoving = false;

        if (Greenfoot.isKeyDown("d")) {
            moveRight();
            animateRunRight();
            isFacingLeft = false;
            isMoving = true;
```

```
        }
        if (Greenfoot.isKeyDown("a")) {
            moveLeft();
            animateRunLeft();
            isFacingLeft = true;
            isMoving = true;
        }
        if (Greenfoot.isKeyDown("space")) {
            jump();
        }
        if (Greenfoot.mousePressed(null)) {
            performAttack();
        }

        if (!isMoving && !isJumping && !isAttacking) {
            setIdleImage();
        }
    }
```

> Memeriksa input dari keyboard (tombol "d" untuk bergerak ke kanan, "a" untuk bergerak ke kiri, "space" untuk melompat, dan klik mouse untuk menyerang). Jika tidak ada input pergerakan, karakter akan kembali ke posisi idle.

```
public void moveRight() {
    if (!isTouchingLeftOrRight(Tiles.class) && !isTouchingLeftOrRight(PlatformTiles.class) && !isTouchingLeftOrRight(Crate1.c
        setLocation(getX() + hSpeed, getY());
    }
    animateRunRight();
}

public void moveLeft() {
    if (!isTouchingLeftOrRight(Tiles.class) && !isTouchingLeftOrRight(PlatformTiles.class) && !isTouchingLeftOrRight(Crate1.c
        setLocation(getX() - hSpeed, getY());
    }
    animateRunLeft();
}

public void fall() {
    detectPlatform();
    setLocation(getX(), getY() + vSpeed);

    if (vSpeed < maxJumpSpeed) {
        vSpeed = maxJumpSpeed;
    } else {
        vSpeed += acceleration;
    }
}
```

Penjelasan :

```
public void moveRight() {
        if (!isTouchingLeftOrRight(Tiles.class) && !isTouchingLeftOrRight(PlatformTiles.class)
&& !isTouchingLeftOrRight(Crate1.class)) {
        setLocation(getX() + hSpeed, getY());
    }
```

```
        animateRunRight();
    }

    public void moveLeft() {
        if (!isTouchingLeftOrRight(Tiles.class) && !isTouchingLeftOrRight(PlatformTiles.class)
&& !isTouchingLeftOrRight(Crate1.class)) {
            setLocation(getX() - hSpeed, getY());
        }
        animateRunLeft();
    }
```

> Karakter hanya akan bergerak jika tidak ada objek (Tiles, PlatformTiles, Crate1) di sisi kiri atau kanan karakter yang menghalangi. Setelah karakter bergerak, animasi untuk berlari ke kanan atau kiri dipanggil.

```
public void fall() {
        detectPlatform();
        setLocation(getX(), getY() + vSpeed);

        if (vSpeed < maxJumpSpeed) {
            vSpeed = maxJumpSpeed;
        } else {
            vSpeed += acceleration;
        }
    }
```

> Memanggil fungsi detectPlatform() untuk memeriksa apakah ada platform di bawah karakter. Kemudian, karakter dipindahkan ke bawah dengan menambah kecepatan vertikal (vSpeed). Jika kecepatan jatuh melebihi batas (maxJumpSpeed), maka kecepatan vertikal tidak akan bertambah lagi. Jika kecepatan jatuh kurang dari batas maksimal, maka kecepatan jatuh akan meningkat sesuai dengan akselerasi yang ditentukan.

```
104     public boolean onGround() {
105         Actor under = getOneObjectAtOffset(0, getImage().getHeight() / 2, Tiles.class);
106         Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, PlatformTiles.class);
107         Actor crateUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Crate1.class);  // Pengecekan Crate1
108         return under != null || platformUnder != null || crateUnder != null;
109     }
110
111     public void checkFall() {
112         if (onGround()) {
113             vSpeed = 0;
114             isJumping = false;
115         } else {
116             isJumping = true;
117             animateJump();
118             fall();
119         }
120     }
121
122     public void jump() {
123         if (onGround()) {
124             vSpeed = -jumpStrength;
125             isJumping = true;
126             animateJump();
127             fall();
128         }
129     }
```

Penjelasan :

```
public boolean onGround() {
      Actor under = getOneObjectAtOffset(0, getImage().getHeight() / 2, Tiles.class);
            Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
PlatformTiles.class);
       Actor crateUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Crate1.class);  //
Pengecekan Crate1
      return under != null || platformUnder != null || crateUnder != null;
   }
```

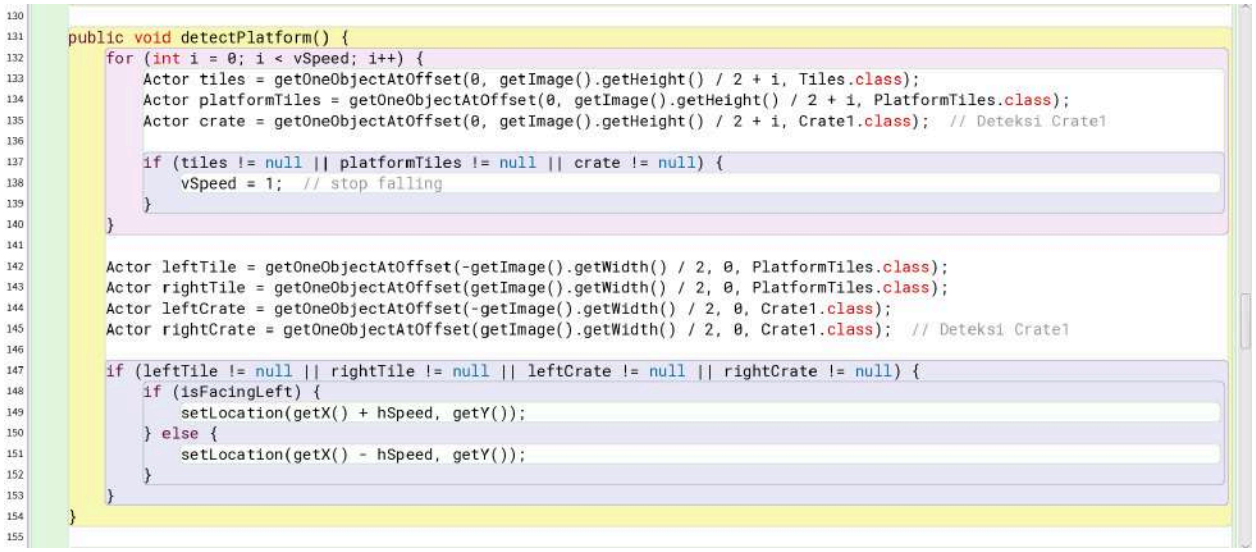
> Memeriksa apakah karakter berada di atas permukaan yang dapat disebut sebagai tanah, seperti platform atau objek lainnya. Dengan menggunakan metode getOneObjectAtOffset(), program memeriksa apakah ada objek yang berada di bawah karakter pada posisi tengah (di bawah gambar karakter). Jika objek tersebut berupa Tiles, PlatformTiles, atau Crate1, maka karakter dianggap sedang berada di tanah atau di atas platform. Fungsi ini mengembalikan nilai boolean true jika ada objek yang ditemukan, menandakan karakter berada di tanah.

```
public void checkFall() {
      if (onGround()) {
         vSpeed = 0;
         isJumping = false;
      } else {
         isJumping = true;
         animateJump();
         fall();
      }
   }
```

> Memeriksa apakah karakter masih berada di tanah atau sudah jatuh. Jika karakter berada di tanah (onGround() mengembalikan true), maka kecepatan jatuh (vSpeed) di-reset menjadi 0, dan status lompat (isJumping) diubah menjadi false. Sebaliknya, jika karakter tidak berada di tanah, maka karakter dianggap sedang jatuh atau melompat, dan status lompat akan diubah menjadi true. Animasi lompat kemudian dipanggil, dan fungsi fall().

```
public void jump() {
      if (onGround()) {
         vSpeed = -jumpStrength;
         isJumping = true;
         animateJump();
         fall();
      }
   }
```

> Diperiksa menggunakan onGround(), jika karakter berada di tanah, kecepatan jatuh (vSpeed) diatur menjadi nilai negatif (-jumpStrength) untuk memberikan dorongan ke atas. Status lompat (isJumping) diubah menjadi true, dan animasi lompat dipanggil. Fungsi fall() juga dipanggil untuk memastikan bahwa karakter akan mulai jatuh setelah mencapai puncak lompatan.



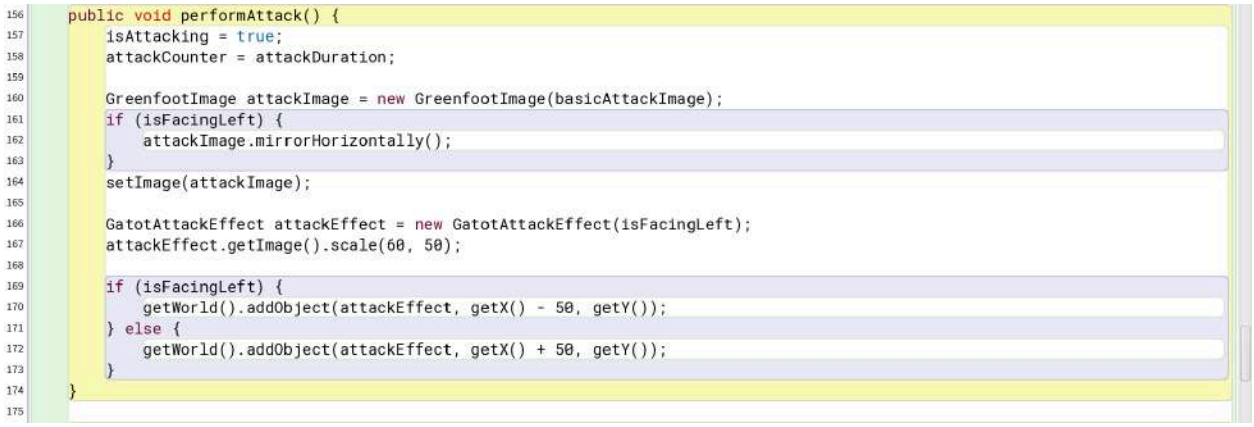
```
130
131     public void detectPlatform() {
132         for (int i = 0; i < vSpeed; i++) {
133             Actor tiles = getOneObjectAtOffset(0, getImage().getHeight() / 2 + i, Tiles.class);
134             Actor platformTiles = getOneObjectAtOffset(0, getImage().getHeight() / 2 + i, PlatformTiles.class);
135             Actor crate = getOneObjectAtOffset(0, getImage().getHeight() / 2 + i, Crate1.class);  // Deteksi Crate1
136
137             if (tiles != null || platformTiles != null || crate != null) {
138                 vSpeed = 1;  // stop falling
139             }
140         }
141
142         Actor leftTile = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, PlatformTiles.class);
143         Actor rightTile = getOneObjectAtOffset(getImage().getWidth() / 2, 0, PlatformTiles.class);
144         Actor leftCrate = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Crate1.class);
145         Actor rightCrate = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Crate1.class);  // Deteksi Crate1
146
147         if (leftTile != null || rightTile != null || leftCrate != null || rightCrate != null) {
148             if (isFacingLeft) {
149                 setLocation(getX() + hSpeed, getY());
150             } else {
151                 setLocation(getX() - hSpeed, getY());
152             }
153         }
154     }
155
```

Penjelasan :

```
public void detectPlatform() {
      for (int i = 0; i < vSpeed; i++) {
         Actor tiles = getOneObjectAtOffset(0, getImage().getHeight() / 2 + i, Tiles.class);
               Actor platformTiles = getOneObjectAtOffset(0, getImage().getHeight() / 2 + i,
PlatformTiles.class);
          Actor crate = getOneObjectAtOffset(0, getImage().getHeight() / 2 + i, Crate1.class);  //
Deteksi Crate1
```

```
        if (tiles != null || platformTiles != null || crate != null) {
            vSpeed = 1;  // stop falling
        }
    }

    Actor leftTile = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, PlatformTiles.class);
    Actor rightTile = getOneObjectAtOffset(getImage().getWidth() / 2, 0, PlatformTiles.class);
    Actor leftCrate = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Crate1.class);
        Actor rightCrate = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Crate1.class);   //
Deteksi Crate1

    if (leftTile != null || rightTile != null || leftCrate != null || rightCrate != null) {
        if (isFacingLeft) {
            setLocation(getX() + hSpeed, getY());
        } else {
            setLocation(getX() - hSpeed, getY());
        }
    }
}
```

> Memeriksa objek di bawah karakter dengan menggunakan getOneObjectAtOffset() untuk mendeteksi Tiles, PlatformTiles, atau Crate1. Jika ada platform atau objek yang terdeteksi, maka kecepatan jatuh (vSpeed) dihentikan (di-set ke 1). Selain itu, fungsi ini juga memeriksa objek di kiri dan kanan karakter untuk memastikan bahwa karakter tidak terjebak di antara objek. Jika platform atau crate terdeteksi di kiri atau kanan, karakter dipindahkan sedikit agar tidak terjebak di antara objek.



```
156     public void performAttack() {
157         isAttacking = true;
158         attackCounter = attackDuration;
159
160         GreenfootImage attackImage = new GreenfootImage(basicAttackImage);
161         if (isFacingLeft) {
162             attackImage.mirrorHorizontally();
163         }
164         setImage(attackImage);
165
166         GatotAttackEffect attackEffect = new GatotAttackEffect(isFacingLeft);
167         attackEffect.getImage().scale(60, 50);
168
169         if (isFacingLeft) {
170             getWorld().addObject(attackEffect, getX() - 50, getY());
171         } else {
172             getWorld().addObject(attackEffect, getX() + 50, getY());
173         }
174     }
175
```

Penjelasan :

```
public void performAttack() {
    isAttacking = true;
```

```
        attackCounter = attackDuration;

        GreenfootImage attackImage = new GreenfootImage(basicAttackImage);
        if (isFacingLeft) {
            attackImage.mirrorHorizontally();
        }
        setImage(attackImage);

        GatotAttackEffect attackEffect = new GatotAttackEffect(isFacingLeft);
        attackEffect.getImage().scale(60, 50);

        if (isFacingLeft) {
            getWorld().addObject(attackEffect, getX() - 50, getY());
        } else {
            getWorld().addObject(attackEffect, getX() + 50, getY());
        }
    }
```

> Ketika dipanggil, status isAttacking diubah menjadi true, dan durasi serangan (attackCounter) diatur sesuai durasi yang ditentukan (attackDuration). Gambar serangan (basicAttackImage) akan digunakan untuk mengganti gambar karakter dengan gambar serangan yang sesuai. Jika karakter menghadap ke kiri, gambar serangan akan dibalik secara horizontal dengan mirrorHorizontally(). Selain itu, efek serangan (GatotAttackEffect) akan ditambahkan ke dunia pada posisi yang sesuai dengan arah karakter.

```
176     public void initAnimationSprites() {
177         for (int i = 0; i < 7; i++) {
178             String filename = "GatotRun" + i + ".png";
179             runRight[i] = new GreenfootImage(filename);
180             runRight[i].scale(110, 130);
181         }
182
183         for (int i = 0; i < 7; i++) {
184             String filename = "GatotRun" + i + ".png";
185             runLeft[i] = new GreenfootImage(filename);
186             runLeft[i].scale(110, 130);
187             runLeft[i].mirrorHorizontally();
188         }
189
190         for (int i = 0; i < 2; i++) {
191             String filename = "GatotJump" + i + ".png";
192             jumpImages[i] = new GreenfootImage(filename);
193             jumpImages[i].scale(90, 130);
194         }
195     }
196
```

Penjelasan :

```
public void initAnimationSprites() {
    for (int i = 0; i < 7; i++) {
        String filename = "GatotRun" + i + ".png";
        runRight[i] = new GreenfootImage(filename);
        runRight[i].scale(110, 130);
```

```
        }

        for (int i = 0; i < 7; i++) {
            String filename = "GatotRun" + i + ".png";
            runLeft[i] = new GreenfootImage(filename);
            runLeft[i].scale(110, 130);
            runLeft[i].mirrorHorizontally();
        }

        for (int i = 0; i < 2; i++) {
            String filename = "GatotJump" + i + ".png";
            jumpImages[i] = new GreenfootImage(filename);
            jumpImages[i].scale(90, 130);
        }
    }
```

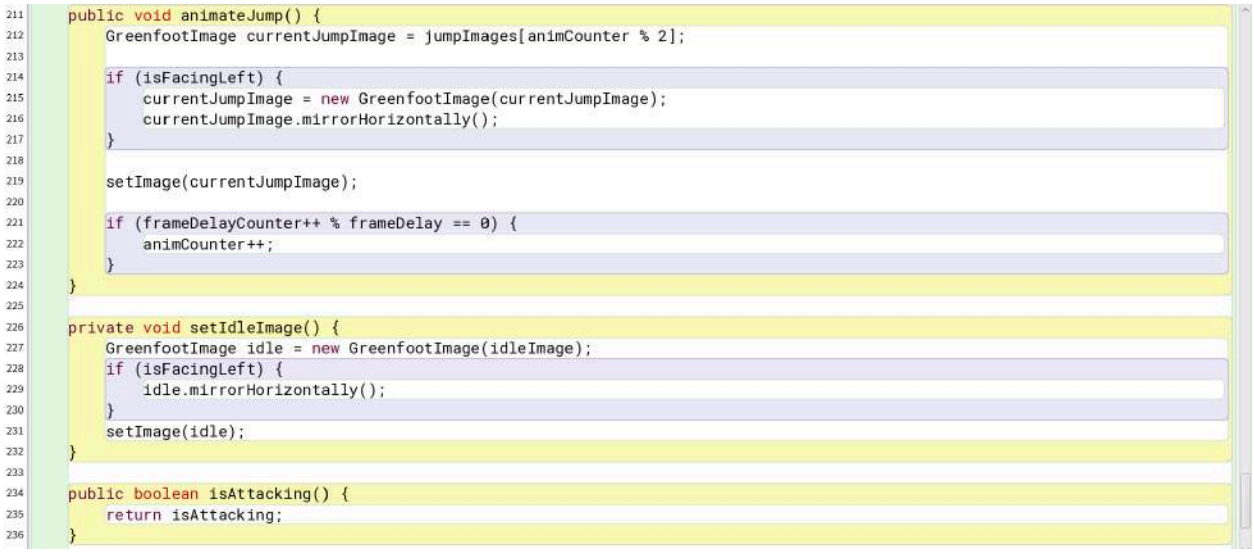
> Memuat gambar-gambar untuk animasi berlari ke kanan (runRight[]) dan ke kiri (runLeft[]). Gambar berlari disusun dalam array dan diperkecil ukurannya untuk menyesuaikan dengan ukuran karakter. Begitu juga gambar lompat (jumpImages[]) yang hanya memiliki dua frame.

```
197     public void animateRunRight() {
198         if (frameDelayCounter++ % frameDelay == 0) {
199             setImage(runRight[animCounter % 7]);
200             animCounter++;
201         }
202     }
203
204     public void animateRunLeft() {
205         if (frameDelayCounter++ % frameDelay == 0) {
206             setImage(runLeft[animCounter % 7]);
207             animCounter++;
208         }
209     }
210
```

Penjelasan :

```
public void animateRunRight() {
        if (frameDelayCounter++ % frameDelay == 0) {
            setImage(runRight[animCounter % 7]);
            animCounter++;
        }
    }

    public void animateRunLeft() {
        if (frameDelayCounter++ % frameDelay == 0) {
            setImage(runLeft[animCounter % 7]);
            animCounter++;
        }
    }
```

> Memeriksa apakah waktu telah cukup (menggunakan frameDelayCounter) untuk melanjutkan ke frame berikutnya. Jika ya, gambar animasi berlari akan berubah sesuai dengan frame yang diinginkan dalam array runRight[] atau runLeft[]. Frame animasi kemudian diganti berdasarkan indeks yang dihitung dengan animCounter, yang akan terus bertambah untuk menciptakan efek animasi berlari.



```
211     public void animateJump() {
212         GreenfootImage currentJumpImage = jumpImages[animCounter % 2];
213
214         if (isFacingLeft) {
215             currentJumpImage = new GreenfootImage(currentJumpImage);
216             currentJumpImage.mirrorHorizontally();
217         }
218
219         setImage(currentJumpImage);
220
221         if (frameDelayCounter++ % frameDelay == 0) {
222             animCounter++;
223         }
224     }
225
226     private void setIdleImage() {
227         GreenfootImage idle = new GreenfootImage(idleImage);
228         if (isFacingLeft) {
229             idle.mirrorHorizontally();
230         }
231         setImage(idle);
232     }
233
234     public boolean isAttacking() {
235         return isAttacking;
236     }
```

Penjelasan :

```
public void animateJump() {
    GreenfootImage currentJumpImage = jumpImages[animCounter % 2];

    if (isFacingLeft) {
       currentJumpImage = new GreenfootImage(currentJumpImage);
       currentJumpImage.mirrorHorizontally();
    }

    setImage(currentJumpImage);

    if (frameDelayCounter++ % frameDelay == 0) {
       animCounter++;
    }
  }
```

> Gambar lompat dipilih dari array jumpImages[] berdasarkan frame saat ini (dihitung dengan animCounter). Jika karakter menghadap ke kiri, gambar lompat dibalik secara horizontal menggunakan mirrorHorizontally().

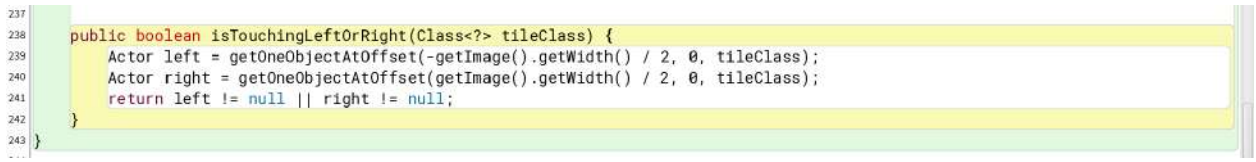
```
private void setIdleImage() {
    GreenfootImage idle = new GreenfootImage(idleImage);
```

```
        if (isFacingLeft) {
            idle.mirrorHorizontally();
        }
        setImage(idle);
    }
```

> menampilkan gambar karakter saat karakter berada dalam keadaan diam (idle). Gambar default karakter (idleImage) akan digunakan, dan jika karakter menghadap ke kiri, gambar akan dibalik menggunakan mirrorHorizontally().

```
public boolean isAttacking() {
        return isAttacking;
    }
```

> Mengembalikan nilai boolean yang menunjukkan apakah karakter sedang melakukan serangan (isAttacking).



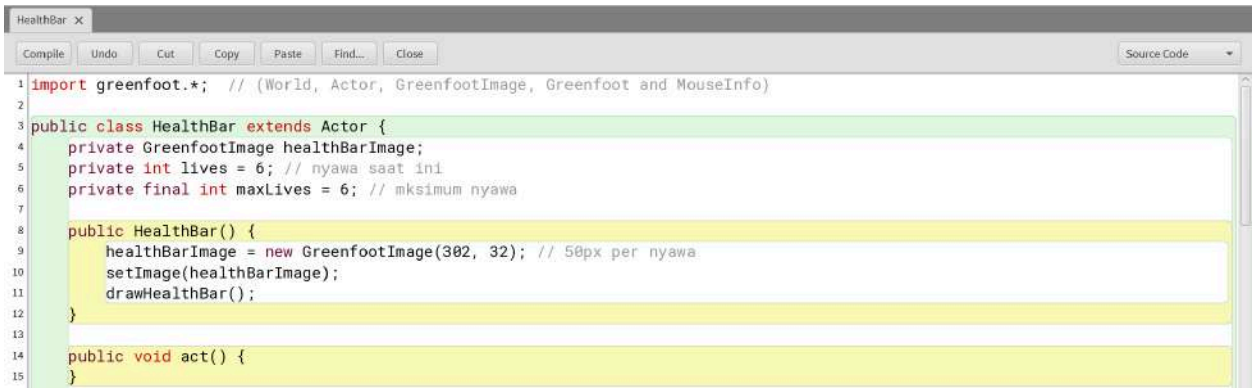
```
237
238     public boolean isTouchingLeftOrRight(Class<?> tileClass) {
239         Actor left = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, tileClass);
240         Actor right = getOneObjectAtOffset(getImage().getWidth() / 2, 0, tileClass);
241         return left != null || right != null;
242     }
243 }
```

Penjelasan :

```
public boolean isTouchingLeftOrRight(Class<?> tileClass) {
        Actor left = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, tileClass);
        Actor right = getOneObjectAtOffset(getImage().getWidth() / 2, 0, tileClass);
        return left != null || right != null;
    }
```

> Dengan menggunakan metode getOneObjectAtOffset(), fungsi ini memeriksa objek pada posisi kiri dan kanan karakter (setengah lebar karakter). Jika ada objek dari kelas yang diberikan (misalnya Tiles, PlatformTiles, atau Crate1), maka fungsi ini mengembalikan true, menandakan bahwa ada objek yang menghalangi gerakan horizontal karakter.

**HealthBar (Sistem nyawa pemain, indikator GameOver)**




Penjelasan :

```
private GreenfootImage healthBarImage;
private int lives = 6; // nyawa saat ini
private final int maxLives = 6; // maksimum nyawa
```

> healthBarImage: Menyimpan gambar untuk tampilan bar kesehatan. Ini digunakan untuk menggambar representasi visual dari nyawa karakter.

> lives: Menyimpan jumlah nyawa saat ini. Nilai awalnya adalah 6, yang berarti karakter memiliki 6 nyawa pada awalnya.

> maxLives: Menyimpan jumlah maksimum nyawa yang bisa dimiliki karakter, yang juga di-set ke 6 di awal.

```
public HealthBar() {
    healthBarImage = new GreenfootImage(302, 32); // 50px per nyawa
    setImage(healthBarImage);
    drawHealthBar();
}
```

> GreenfootImage(302, 32): Membuat objek gambar untuk bar kesehatan dengan lebar 302 pixel dan tinggi 32 pixel. Lebar ini cukup untuk menampilkan 6 nyawa dengan setiap nyawa diwakili oleh 50 pixel.

> setImage(healthBarImage): Mengatur gambar dari aktor (HealthBar) menjadi healthBarImage yang baru saja dibuat.

> drawHealthBar(): Memanggil metode drawHealthBar() untuk menggambar bar kesehatan pada gambar yang telah diatur.

```
    public void decreaseLife() {
        if (lives > 0) {
            lives--;
            drawHealthBar();
        }

        if (lives <= 0) {
            BaseWorld world = (BaseWorld) getWorld();
            // Hentikan musik jika world memiliki musik
            if (world instanceof FiLLiT1) {
                ((FiLLiT1) world).stopMusic();
            } else if (world instanceof Level1) {
                ((Level1) world).stopMusic();
            } else if (world instanceof NumQuest1) {
                ((NumQuest1) world).stopMusic();
            } else if (world instanceof Labyrntine1) {
                ((Labyrntine1) world).stopMusic();
            } else if (world instanceof BossFight1) {
                ((BossFight1) world).stopMusic();
            } else if (world instanceof Level2) {
                ((Level2) world).stopMusic();
            } else if (world instanceof FiLLiT2) {
                ((FiLLiT2) world).stopMusic();
            } else if (world instanceof NumQuest2) {
                ((NumQuest2) world).stopMusic();
            } else if (world instanceof Labyrntine2) {
                ((Labyrntine2) world).stopMusic();
            } else if (world instanceof BossFight2) {
                ((BossFight2) world).stopMusic();
            } else if (world instanceof Level3) {
                ((Level3) world).stopMusic();
            } else if (world instanceof FiLLiT3) {
                ((FiLLiT3) world).stopMusic();
            } else if (world instanceof NumQuest3) {
                ((NumQuest3) world).stopMusic();
            } else if (world instanceof Labyrntine3) {
                ((Labyrntine3) world).stopMusic();
            } else if (world instanceof BossFight3) {
                ((BossFight3) world).stopMusic();
            } else if (world instanceof Level4) {
                ((Level4) world).stopMusic();
            } else if (world instanceof FiLLiT4) {
                ((FiLLiT4) world).stopMusic();
            } else if (world instanceof NumQuest4) {
                ((NumQuest4) world).stopMusic();
            } else if (world instanceof Labyrntine4) {
                ((Labyrntine4) world).stopMusic();
            } else if (world instanceof BossFight4) {
                ((BossFight4) world).stopMusic();
            } else if (world instanceof Level5) {
                ((Level5) world).stopMusic();
            } else if (world instanceof FiLLiT5) {
                ((FiLLiT5) world).stopMusic();
            } else if (world instanceof NumQuest5) {
                ((NumQuest5) world).stopMusic();
            } else if (world instanceof Labyrntine5) {
                ((Labyrntine5) world).stopMusic();
            } else if (world instanceof BossFight5) {
                ((BossFight5) world).stopMusic();
            }
            int finalScore = world.getCounter().getScore();
            Greenfoot.setWorld(new Game_Over(finalScore)); // Pindah ke Game_Over world
        }
    }


    public void increaseLife() {
        if (lives < maxLives) {
            lives++;
            drawHealthBar();
        }
    }

    public boolean canIncreaseLife() {
        return lives < maxLives; // return true jika HealthBar belum penuh
    }

```

Penjelasan :

```
public void decreaseLife() {
      if (lives > 0) {
         lives--;
         drawHealthBar();
      }

      if (lives <= 0) {
         BaseWorld world = (BaseWorld) getWorld();
         // Hentikan musik jika world memiliki musik
         if (world instanceof FiLLiT1) {
            ((FiLLiT1) world).stopMusic();
         } else if
                  …..
         }
         int finalScore = world.getCounter().getScore();
         Greenfoot.setWorld(new Game_Over(finalScore)); // Pindah ke Game_Over world
      }
   }
```

> Jika nyawa lebih dari 0, nilai lives dikurangi satu dan tampilan bar kesehatan diperbarui dengan memanggil drawHealthBar().

> Jika nyawa habis (nyawa <= 0), program menghentikan musik yang sedang diputar di dunia saat ini, mengambil nilai skor akhir, dan kemudian berpindah ke dunia Game_Over yang menunjukkan hasil akhir permainan.

```
public void increaseLife() {
      if (lives < maxLives) {
         lives++;
         drawHealthBar();
      }
   }
```

> Jika jumlah nyawa belum mencapai batas maksimum (maxLives), maka lives akan bertambah satu dan bar kesehatan akan diperbarui dengan memanggil drawHealthBar().

```
public boolean canIncreaseLife() {
      return lives < maxLives; // return true jika HealthBar belum penuh
   }
```

> Memeriksa apakah nyawa karakter bisa ditambah. Jika lives kurang dari maxLives, fungsi ini akan mengembalikan true, yang berarti nyawa bisa ditambah. Jika lives sudah mencapai maxLives, maka mengembalikan false.

```
    private void drawHealthBar() {
        healthBarImage.setColor(Color.BLACK);
        healthBarImage.fillRect(0, 0, 301, 31); // panjang sesuai jumlah lives

        for (int i = 0; i < lives; i++) {
            healthBarImage.setColor(Color.GREEN);
            healthBarImage.fillRect(1 + i * 50, 1, 48, 30); // 50 pixel untuk setiap nyawa
        }
    }
}
```

Penjelasan :

```
private void drawHealthBar() {
      healthBarImage.setColor(Color.BLACK);
      healthBarImage.fillRect(0, 0, 301, 31); // panjang sesuai jumlah lives

      for (int i = 0; i < lives; i++) {
         healthBarImage.setColor(Color.GREEN);
         healthBarImage.fillRect(1 + i * 50, 1, 48, 30); // 50 pixel untuk setiap nyawa
      }
   }
```

> Bar kesehatan pertama-tama digambar dengan latar belakang hitam (setColor(Color.BLACK) dan fillRect(0, 0, 301, 31)). Untuk setiap nyawa (for (int i = 0; i < lives; i++)), sebuah kotak hijau (berwarna Color.GREEN) digambar di posisi tertentu untuk mewakili satu nyawa. Setiap kotak ini memiliki lebar 50 pixel, dengan sedikit ruang di antara setiap kotak (diatur dengan offset i * 50).

**Heart (Item untuk menambah HealthBar apabila diperlukan dan disentuh oleh Gatot_StandBy)**

Heart

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Heart extends Actor {
    public void act() {
        checkTouchByGatot();
    }

    private void checkTouchByGatot() {
        // detek sentuhan
        Gatot_StandBy gatot = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);

        if (gatot != null) {
            // akses dunia
            BaseWorld world = (BaseWorld) getWorld();
            HealthBar healthBar = world.getHealthBar();

            if (healthBar != null && healthBar.canIncreaseLife()) {
                // +health
                healthBar.increaseLife();
                world.removeObject(this); // remove health
            }
        }
    }
}
```

Penjelasan :

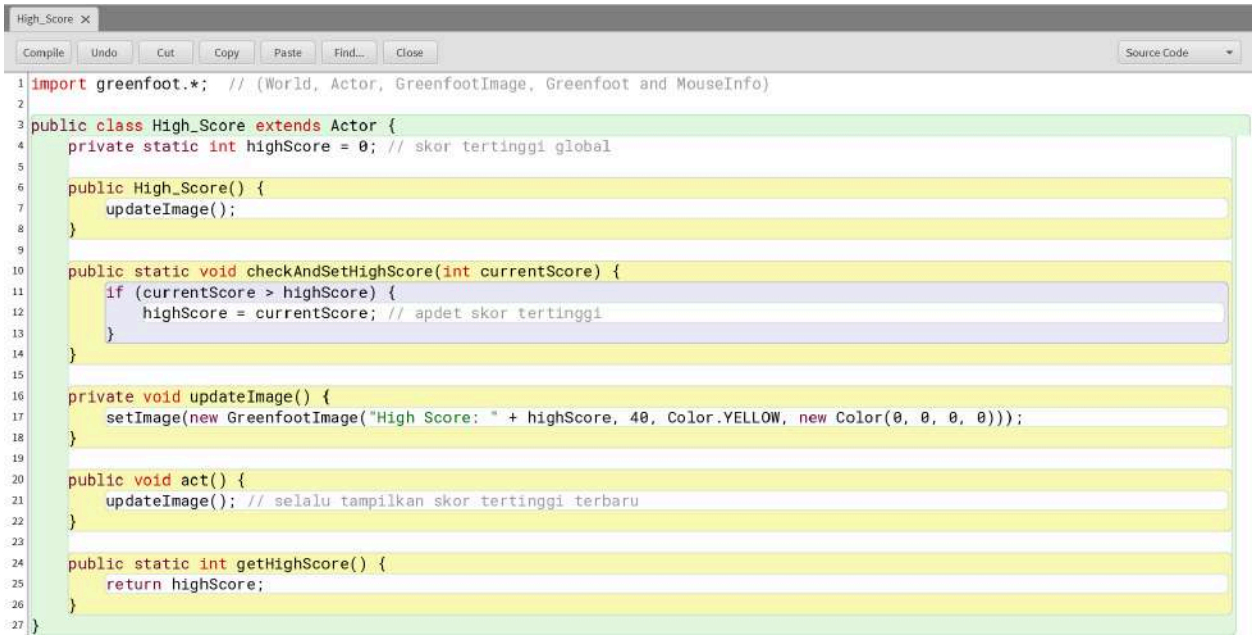
```
public void act() {
```

```
        checkTouchByGatot();
    }
```
> Memanggil checkTouchByGatot().

```
private void checkTouchByGatot() {
        // detek sentuhan
        Gatot_StandBy gatot = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);

        if (gatot != null) {
            // akses dunia
            BaseWorld world = (BaseWorld) getWorld();
            HealthBar healthBar = world.getHealthBar();

            if (healthBar != null && healthBar.canIncreaseLife()) {
                // +health
                healthBar.increaseLife();
                world.removeObject(this); // remove health
            }
        }
    }
```
> Jika Heart bersentuhan dengan Gatot_StandBy, maka:

- Mengakses objek HealthBar di dunia.
- Jika nyawa bisa ditambah (canIncreaseLife()), maka menambah nyawa dengan increaseLife().
- Menghapus objek Heart dari dunia menggunakan removeObject(this) setelah interaksi.

**High_Score**



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class High_Score extends Actor {
    private static int highScore = 0; // skor tertinggi global

    public High_Score() {
        updateImage();
    }

    public static void checkAndSetHighScore(int currentScore) {
        if (currentScore > highScore) {
            highScore = currentScore; // apdet skor tertinggi
        }
    }

    private void updateImage() {
        setImage(new GreenfootImage("High Score: " + highScore, 40, Color.YELLOW, new Color(0, 0, 0, 0)));
    }

    public void act() {
        updateImage(); // selalu tampilkan skor tertinggi terbaru
    }

    public static int getHighScore() {
        return highScore;
    }
}
```

Penjelasan :

> Kelas High_Score di Greenfoot digunakan untuk menampilkan dan memperbarui skor tertinggi dalam permainan. Variabel highScore menyimpan skor tertinggi secara global. Metode checkAndSetHighScore(int currentScore) digunakan untuk memeriksa apakah skor saat ini lebih tinggi dari highScore, dan jika ya, memperbarui nilai skor tertinggi. Metode updateImage() mengubah gambar objek untuk menampilkan skor tertinggi terbaru dengan warna kuning pada latar belakang transparan. Di dalam metode act(), gambar diperbarui secara terus-menerus agar skor tertinggi selalu ditampilkan dengan benar. Metode getHighScore() memungkinkan akses untuk mendapatkan nilai skor tertinggi.

**Name**

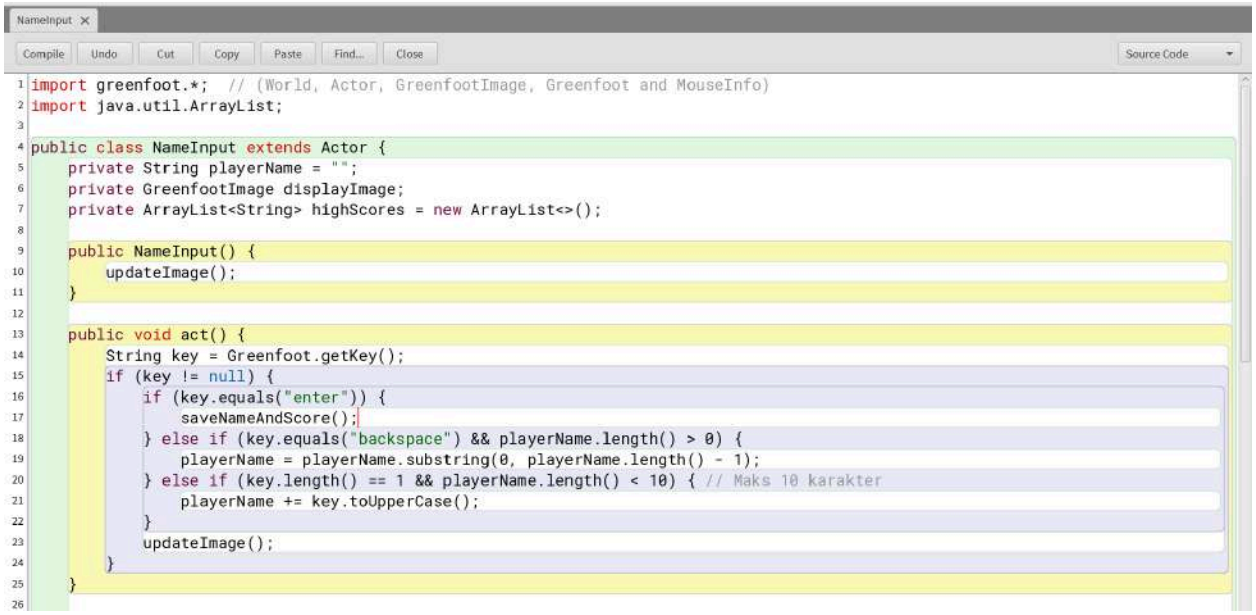
```
import greenfoot.*;

public class Name extends Actor {
    private GreenfootImage displayImage;

    public Name(String text) {
        displayImage = new GreenfootImage(text, 50, Color.WHITE, new Color(0, 0, 0, 0));
        setImage(displayImage);
    }
}
```

Penjelasan :

Name(String text) menerima parameter text, yang berupa teks yang ingin ditampilkan. Objek GreenfootImage dibuat dengan teks tersebut, ukuran font 50, warna teks putih, dan latar

belakang transparan (RGBA: 0, 0, 0, 0). Gambar yang dihasilkan ini kemudian diatur sebagai gambar objek menggunakan setImage(displayImage), sehingga teks akan muncul pada posisi objek Name di dunia.

**NameInput**




```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.ArrayList;

public class NameInput extends Actor {
    private String playerName = "";
    private GreenfootImage displayImage;
    private ArrayList<String> highScores = new ArrayList<>();

    public NameInput() {
        updateImage();
    }

    public void act() {
        String key = Greenfoot.getKey();
        if (key != null) {
            if (key.equals("enter")) {
                saveNameAndScore();
            } else if (key.equals("backspace") && playerName.length() > 0) {
                playerName = playerName.substring(0, playerName.length() - 1);
            } else if (key.length() == 1 && playerName.length() < 10) { // Maks 10 karakter
                playerName += key.toUpperCase();
            }
            updateImage();
        }
    }

```

Penjelasan :

```
private String playerName = "";
   private GreenfootImage displayImage;
   private ArrayList<String> highScores = new ArrayList<>();

   public NameInput() {
      updateImage();
   }
```

> playerName: Menyimpan nama pemain yang dimasukkan.
> displayImage: Gambar yang digunakan untuk menampilkan teks input.
> highScores: Array list untuk menyimpan skor tertinggi.
> NameInput() memanggil metode updateImage() untuk memperbarui gambar dengan teks input yang ada.

```
public void act() {
      String key = Greenfoot.getKey();
      if (key != null) {
         if (key.equals("enter")) {
            saveNameAndScore();
         } else if (key.equals("backspace") && playerName.length() > 0) {
            playerName = playerName.substring(0, playerName.length() - 1);
```

```
        } else if (key.length() == 1 && playerName.length() < 10) { // Maks 10 karakter
            playerName += key.toUpperCase();
        }
        updateImage();
    }
}
```

> Mengambil input dari pemain melalui Greenfoot.getKey().
> Jika tombol "enter" ditekan, nama dan skor disimpan menggunakan metode saveNameAndScore().
> Jika tombol "backspace" ditekan dan panjang nama lebih dari 0, menghapus karakter terakhir dari nama pemain.
> Jika input adalah satu karakter dan panjang nama kurang dari 10, karakter tersebut ditambahkan ke nama pemain.

```
27      private void updateImage() {
28          displayImage = new GreenfootImage("Enter Name: " + playerName, 30, Color.WHITE, new Color(0, 0, 0, 0));
29          setImage(displayImage);
30      }
31
32      private void saveNameAndScore() {
33          BaseWorld world = (BaseWorld) getWorld();
34          Counter counter = world.getCounter();
35          String entry = playerName + " - " + counter.getScore();
36          highScores.add(entry);
37          playerName = ""; // Reset name input
38          counter.setGlobalScore(0); // Reset score
39          displayHighScores();
40      }
41
42      private void displayHighScores() {
43          BaseWorld world = (BaseWorld) getWorld();
44          int startY = 100;
45          for (String entry : highScores) {
46              Name nameDisplay = new Name(entry);
47              world.addObject(nameDisplay, 640, startY);
48              startY += 30; // Jarak antar baris
49          }
50          getWorld().removeObject(this); // Hapus input field setelah selesai
51      }
52  }
```

Penjelasan :

```
private void updateImage() {
        displayImage = new GreenfootImage("Enter Name: " + playerName, 30, Color.WHITE, new Color(0, 0, 0, 0));
    setImage(displayImage);
}
```

> Memperbarui gambar objek dengan menampilkan teks "Enter Name: " diikuti dengan nama pemain yang sedang dimasukkan, dengan ukuran font 30 dan teks berwarna putih.

```
private void saveNameAndScore() {
    BaseWorld world = (BaseWorld) getWorld();
    Counter counter = world.getCounter();
    String entry = playerName + " - " + counter.getScore();
```

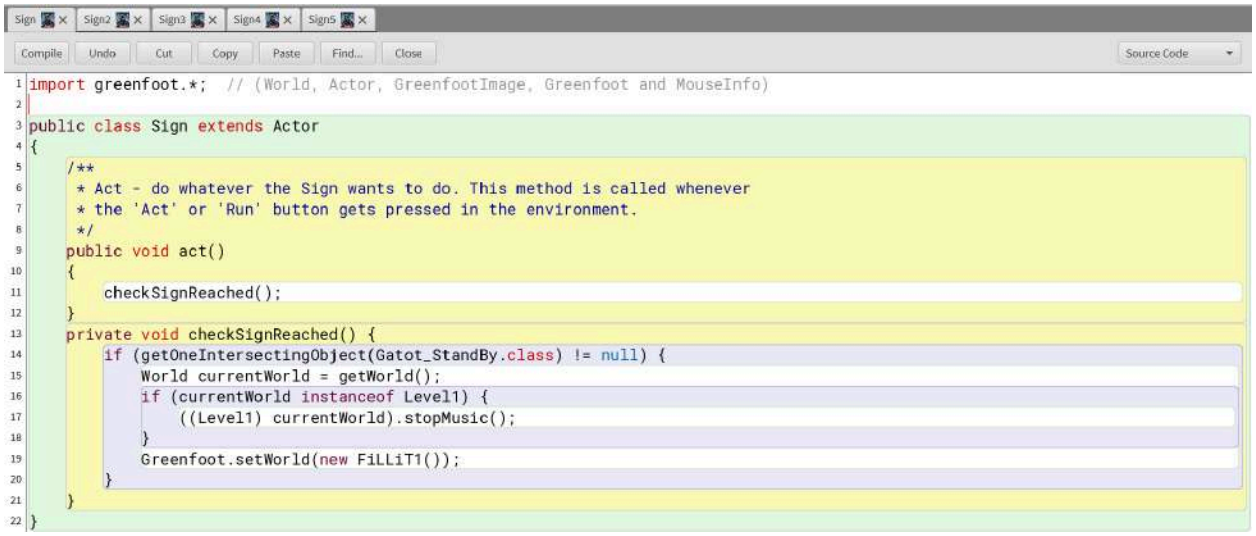
```
        highScores.add(entry);
        playerName = ""; // Reset name input
        counter.setGlobalScore(0); // Reset score
        displayHighScores();
    }
```
> Mengambil objek Counter dari dunia (BaseWorld) untuk mendapatkan skor saat ini.
> Menambahkan entri baru ke highScores dalam format "Nama - Skor".
> Mengosongkan playerName dan mengatur skor global menjadi 0.
> Memanggil displayHighScores() untuk menampilkan daftar skor tertinggi.

```
private void displayHighScores() {
        BaseWorld world = (BaseWorld) getWorld();
        int startY = 100;
        for (String entry : highScores) {
            Name nameDisplay = new Name(entry);
            world.addObject(nameDisplay, 640, startY);
            startY += 30; // Jarak antar baris
        }
        getWorld().removeObject(this); // Hapus input field setelah selesai
    }
```
> Menampilkan setiap entri dari highScores pada layar, dengan menambahkan objek Name untuk setiap skor tertinggi, disusun vertikal dengan jarak antar baris 30 piksel.
> Setelah semua skor tertinggi ditampilkan, objek NameInput dihapus dari world.

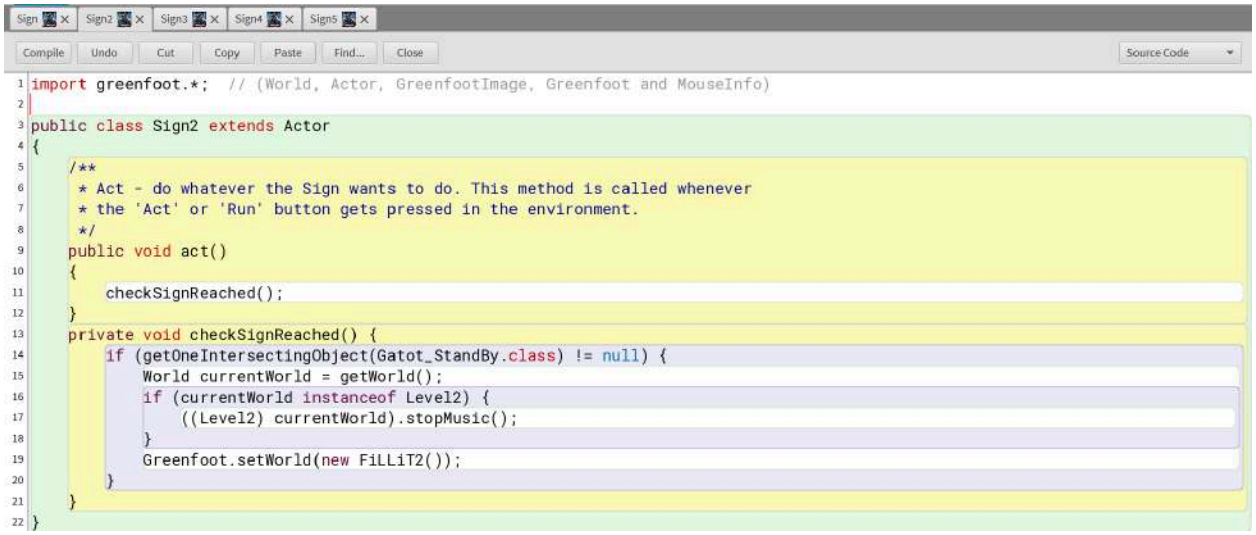
**Sign (Garis Finish pada world “Level”)**

Terdapat 5 buah Sign yang berfungsi sebagai garis finish pada setiap world “Level”, dimana apabila Gatot_StandBy telah berhasil menyentuh Sign yang biasanya berada pada akhir dari width asli world tersebut, maka musik pada world tersebut akan dihentikan dan akan berpindah ke world “FiLLiT” sesuai dengan world “Level” yang sedang berjalan.

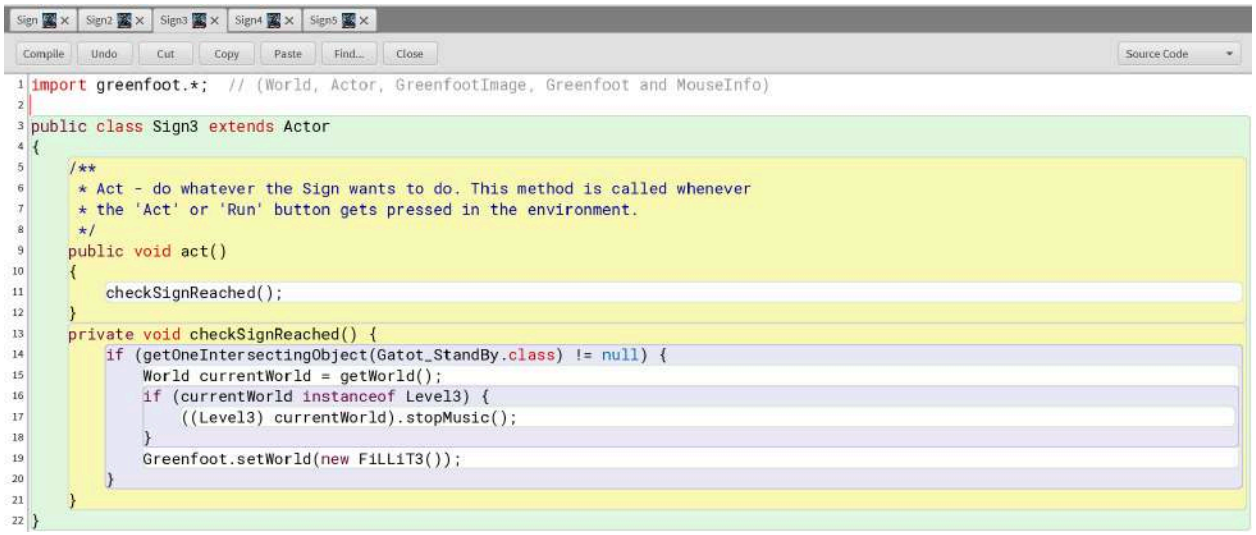
**Sign**


Sign
Sign2
Sign3
Sign4
Sign5
Compile
Undo
Cut
Copy
Paste
Find...
Close
Source Code
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
public class Sign extends Actor
{
/**
* Act - do whatever the Sign wants to do. This method is called whenever
* the 'Act' or 'Run' button gets pressed in the environment.
*/
public void act()
{
checkSignReached();
}
private void checkSignReached() {
if (getOneIntersectingObject(Gatot_StandBy.class) != null) {
World currentWorld = getWorld();
if (currentWorld instanceof Level1) {
((Level1) currentWorld).stopMusic();
}
Greenfoot.setWorld(new FiLLiT1());
}
}
}

## Sign2

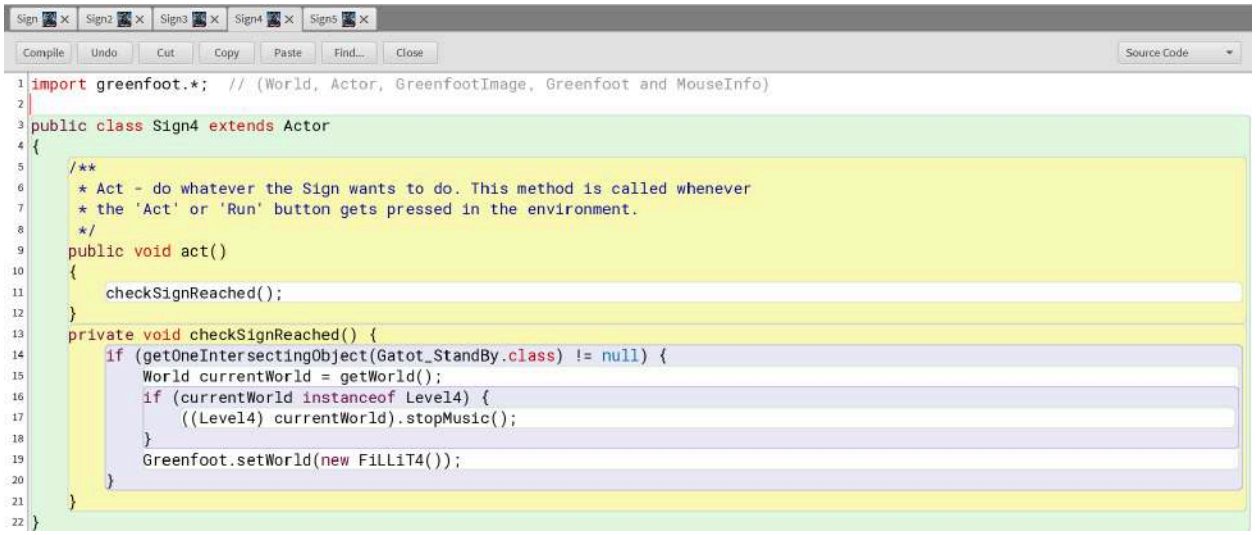

Sign
Sign2
Sign3
Sign4
Sign5
Compile
Undo
Cut
Copy
Paste
Find...
Close
Source Code
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
public class Sign2 extends Actor
{
/**
* Act - do whatever the Sign wants to do. This method is called whenever
* the 'Act' or 'Run' button gets pressed in the environment.
*/
public void act()
{
checkSignReached();
}
private void checkSignReached() {
if (getOneIntersectingObject(Gatot_StandBy.class) != null) {
World currentWorld = getWorld();
if (currentWorld instanceof Level2) {
((Level2) currentWorld).stopMusic();
}
Greenfoot.setWorld(new FiLLiT2());
}
}
}

## Sign3




```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Sign3 extends Actor
{
    /**
     * Act - do whatever the Sign wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkSignReached();
    }
    private void checkSignReached() {
        if (getOneIntersectingObject(Gatot_StandBy.class) != null) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Level3) {
                ((Level3) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new FiLLiT3());
        }
    }
}
```

## Sign4




```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Sign4 extends Actor
{
    /**
     * Act - do whatever the Sign wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkSignReached();
    }
    private void checkSignReached() {
        if (getOneIntersectingObject(Gatot_StandBy.class) != null) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Level4) {
                ((Level4) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new FiLLiT4());
        }
    }
}
```

## Sign5

Sign | Sign2 | Sign3 | Sign4 | Sign5

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Sign5 extends Actor
{
    /**
     * Act - do whatever the Sign wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkSignReached();
    }
    private void checkSignReached() {
        if (getOneIntersectingObject(Gatot_StandBy.class) != null) {
            World currentWorld = getWorld();
            if (currentWorld instanceof Level5) {
                ((Level5) currentWorld).stopMusic();
            }
            Greenfoot.setWorld(new FiLLiT5());
        }
    }
}
```

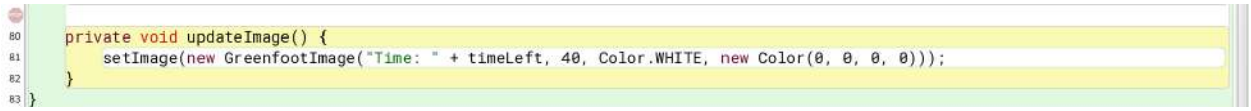
## Timer

Timer

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Timer extends Actor {
    private int timeLeft;
    private long lastUpdateTime;

    public Timer() {
        this.timeLeft = 300;
        updateImage();
    }

    public void act() {
        if (System.currentTimeMillis() - lastUpdateTime >= 1000) {
            lastUpdateTime = System.currentTimeMillis();
            if (timeLeft > 0) {
                timeLeft--;
            }
            updateImage();

            if (timeLeft == 0) {
                int finalScore = Counter.getGlobalScore(); // Ambil skor global
                BaseWorld world = (BaseWorld) getWorld();
                // Hentikan musik jika world memiliki musik
                if (world instanceof FiLLiT1) {
                    ((FiLLiT1) world).stopMusic();
                } else if (world instanceof Level1) {
                    ((Level1) world).stopMusic();
                } else if (world instanceof NumQuest1) {
                    ((NumQuest1) world).stopMusic();
                } else if (world instanceof Labyrntine1) {
                    ((Labyrntine1) world).stopMusic();
                } else if (world instanceof BossFight1) {
                    ((BossFight1) world).stopMusic();
                } else if (world instanceof Level2) {
                    ((Level2) world).stopMusic();
                } else if (world instanceof FiLLiT2) {
                    ((FiLLiT2) world).stopMusic();
                } else if (world instanceof NumQuest2) {
                    ((NumQuest2) world).stopMusic();
                } else if (world instanceof Labyrntine2) {
                    ((Labyrntine2) world).stopMusic();
                } else if (world instanceof BossFight2) {
                    ((BossFight2) world).stopMusic();
                } else if (world instanceof Level3) {
                    ((Level3) world).stopMusic();
                } else if (world instanceof FiLLiT3) {
                    ((FiLLiT3) world).stopMusic();
                } else if (world instanceof NumQuest3) {
                    ((NumQuest3) world).stopMusic();
                } else if (world instanceof Labyrntine3) {
                    ((Labyrntine3) world).stopMusic();
                } else if (world instanceof BossFight3) {
                    ((BossFight3) world).stopMusic();
```

```
            } else if (world instanceof Level4) {
                ((Level4) world).stopMusic();
            } else if (world instanceof FiLLiT4) {
                ((FiLLiT4) world).stopMusic();
            } else if (world instanceof NumQuest4) {
                ((NumQuest4) world).stopMusic();
            } else if (world instanceof Labyrntine4) {
                ((Labyrntine4) world).stopMusic();
            } else if (world instanceof BossFight4) {
                ((BossFight4) world).stopMusic();
            } else if (world instanceof Level5) {
                ((Level5) world).stopMusic();
            } else if (world instanceof FiLLiT5) {
                ((FiLLiT5) world).stopMusic();
            } else if (world instanceof NumQuest5) {
                ((NumQuest5) world).stopMusic();
            } else if (world instanceof Labyrntine5) {
                ((Labyrntine5) world).stopMusic();
            } else if (world instanceof BossFight5) {
                ((BossFight5) world).stopMusic();
            }
            Greenfoot.setWorld(new Game_Over(finalScore));
        }
    }
}
```



```
    private void updateImage() {
        setImage(new GreenfootImage("Time: " + timeLeft, 40, Color.WHITE, new Color(0, 0, 0, 0)));
    }
}
```

Penjelasan :

```
private int timeLeft;
private long lastUpdateTime;
```

> private int timeLeft; : Variabel untuk menyimpan jumlah waktu yang tersisa dalam hitungan detik. Awalnya diatur di konstruktor.

> private long lastUpdateTime; : Variabel untuk mencatat waktu saat terakhir kali timer diperbarui, menggunakan milidetik dari System.currentTimeMillis().

```
public Timer() {
      this.timeLeft = 300;
      updateImage();
   }
```

> Membuat timeLeft ke nilai awal (300 detik atau 5 menit) dan memanggil metode updateImage() untuk menampilkan waktu awal pada layar.

```
public void act() {
      if (System.currentTimeMillis() - lastUpdateTime >= 1000) {
         lastUpdateTime = System.currentTimeMillis();
         if (timeLeft > 0) {
            timeLeft--;
         }
         updateImage();

         if (timeLeft == 0) {
            int finalScore = Counter.getGlobalScore(); // Ambil skor global
```

```
            BaseWorld world = (BaseWorld) getWorld();
            // Hentikan musik jika world memiliki musik
            if (world instanceof FiLLiT1) {
               ((FiLLiT1) world).stopMusic();
            } else if
                  …..
            }
            Greenfoot.setWorld(new Game_Over(finalScore));
        }
    }
}
```

> Perhitungan Timer yang mengecek apakah 1 detik telah berlalu dengan membandingkan waktu saat ini dengan lastUpdateTime. Jika iya maka memperbarui lastUpdateTime ke waktu saat ini. Mengurangi timeLeft sebanyak 1 jika waktu belum habis. Memanggil updateImage() untuk memperbarui tampilan timer.

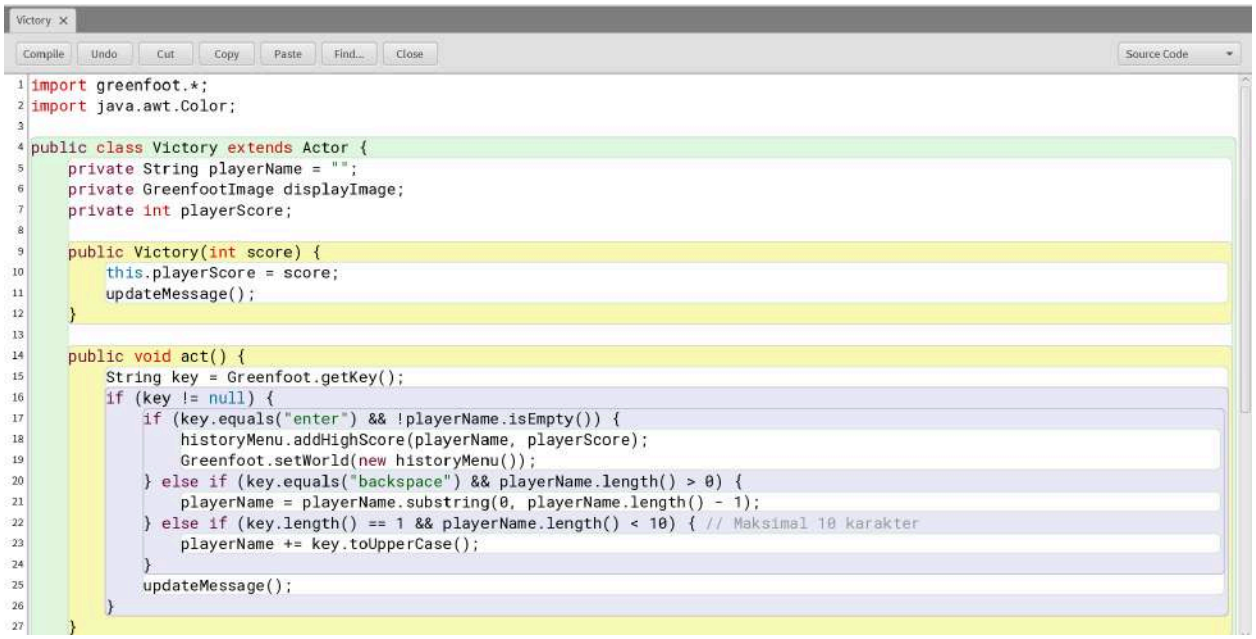
> Ketika Timer Habis

Jika timeLeft mencapai nol: Skor akhir diambil dari Counter.getGlobalScore() (metode yang mengelola skor global). Mengakses dunia (BaseWorld) tempat timer berada dan memanggil metode stopMusic() dari kelas dunia tertentu untuk menghentikan musik. Setelah musik dihentikan, dunia diganti ke Game_Over menggunakan Greenfoot.setWorld(), dan skor akhir diteruskan ke layar game over.

```
private void updateImage() {
        setImage(new GreenfootImage("Time: " + timeLeft, 40, Color.WHITE, new Color(0, 0, 0, 0)));
    }
```

> Membuat dan menetapkan gambar yang menampilkan waktu tersisa (timeLeft) dengan format "Time: [nilai]" menggunakan warna teks putih. Gambar ini diperbarui setiap detik untuk mencerminkan perubahan waktu.

**Victory**




```java
import greenfoot.*;
import java.awt.Color;

public class Victory extends Actor {
    private String playerName = "";
    private GreenfootImage displayImage;
    private int playerScore;

    public Victory(int score) {
        this.playerScore = score;
        updateMessage();
    }

    public void act() {
        String key = Greenfoot.getKey();
        if (key != null) {
            if (key.equals("enter") && !playerName.isEmpty()) {
                historyMenu.addHighScore(playerName, playerScore);
                Greenfoot.setWorld(new historyMenu());
            } else if (key.equals("backspace") && playerName.length() > 0) {
                playerName = playerName.substring(0, playerName.length() - 1);
            } else if (key.length() == 1 && playerName.length() < 10) { // Maksimal 10 karakter
                playerName += key.toUpperCase();
            }
            updateMessage();
        }
    }
```

Penjelasan :

```
private String playerName = "";
private GreenfootImage displayImage;
private int playerScore;
```

> private String playerName = ""; : Variabel untuk menyimpan nama pemain yang dimasukkan melalui input keyboard.

> private GreenfootImage displayImage; : Variabel untuk menyimpan gambar tampilan (teks dan latar) yang akan diperlihatkan di layar.

> private int playerScore; : Variabel untuk menyimpan skor pemain yang diteruskan melalui konstruktor.

```
public Victory(int score) {
      this.playerScore = score;
      updateMessage();
   }
```

> Konstruktor menerima skor pemain (score) sebagai parameter, lalu menginisialisasi playerScore dengan nilai tersebut. Selanjutnya, memanggil updateMessage() untuk menampilkan pesan kemenangan.

```
public void act() {
      String key = Greenfoot.getKey();
      if (key != null) {
         if (key.equals("enter") && !playerName.isEmpty()) {
```

```
            historyMenu.addHighScore(playerName, playerScore);
            Greenfoot.setWorld(new historyMenu());
        } else if (key.equals("backspace") && playerName.length() > 0) {
            playerName = playerName.substring(0, playerName.length() - 1);
        } else if (key.length() == 1 && playerName.length() < 10) { // Maksimal 10 karakter
            playerName += key.toUpperCase();
        }
        updateMessage();
    }
}
```

> Jika tombol Enter ditekan dan playerName tidak kosong:

- Nama dan skor pemain disimpan ke dalam menu skor melalui historyMenu.addHighScore(playerName, playerScore).
- Berpindah ke dunia historyMenu menggunakan Greenfoot.setWorld(new historyMenu()).

> Jika tombol Backspace ditekan:

- Menghapus satu karakter terakhir dari playerName, asalkan panjang playerName lebih dari 0.

> Jika karakter tunggal ditekan:

- Menambahkan karakter tersebut ke playerName, diubah menjadi huruf besar, selama panjangnya kurang dari 10 karakter.

```
29      private void updateMessage() {
30          displayImage = new GreenfootImage(600, 200);
31          displayImage.setColor(greenfoot.Color.BLACK);
32          displayImage.fill();
33          displayImage.setColor(greenfoot.Color.RED);
34          displayImage.setFont(displayImage.getFont().deriveFont(50f));
35          displayImage.drawString("Victory!", 150, 50);
36
37          displayImage.setColor(greenfoot.Color.WHITE);
38          displayImage.setFont(displayImage.getFont().deriveFont(30f));
39          displayImage.drawString("Enter Name: " + playerName, 100, 120);
40          displayImage.drawString("Score: " + playerScore, 100, 170);
41
42          setImage(displayImage);
43      }
44  }
```

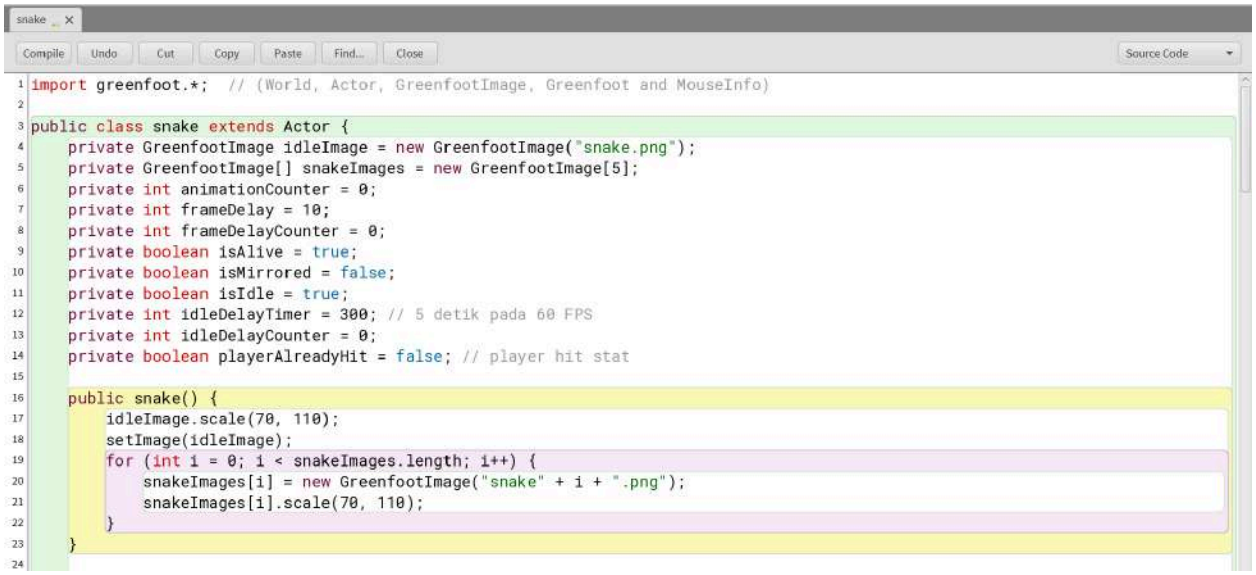
Penjelasan :

```
private void updateMessage() {
    displayImage = new GreenfootImage(600, 200);
    displayImage.setColor(greenfoot.Color.BLACK);
    displayImage.fill();
    displayImage.setColor(greenfoot.Color.RED);
    displayImage.setFont(displayImage.getFont().deriveFont(50f));
    displayImage.drawString("Victory!", 150, 50);

    displayImage.setColor(greenfoot.Color.WHITE);
    displayImage.setFont(displayImage.getFont().deriveFont(30f));
```

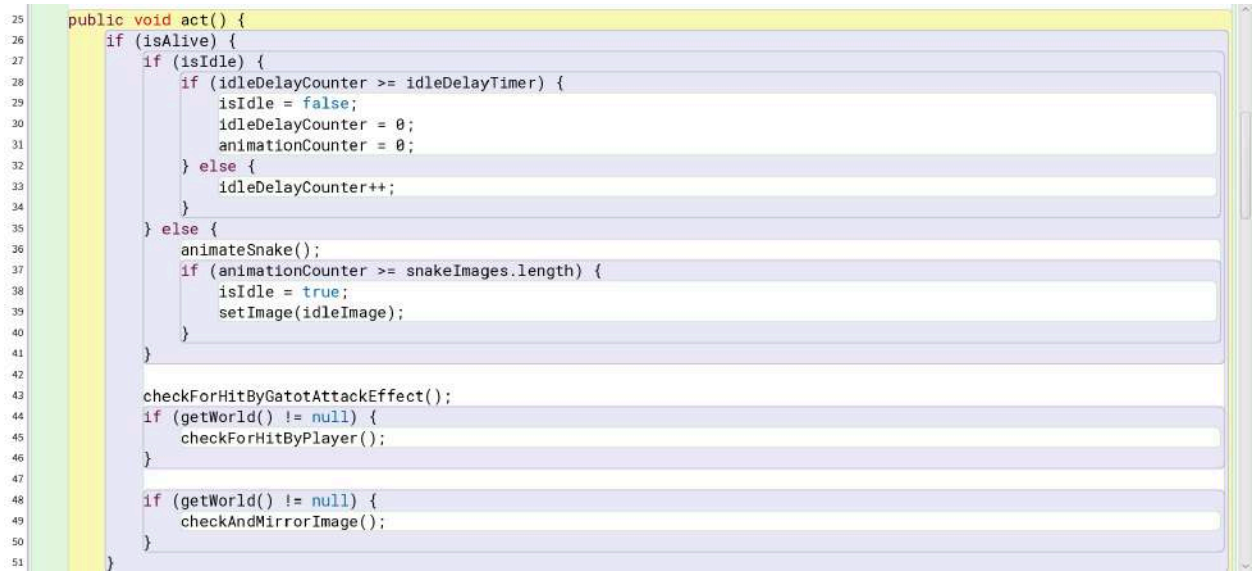
```
        displayImage.drawString("Enter Name: " + playerName, 100, 120);
        displayImage.drawString("Score: " + playerScore, 100, 170);

        setImage(displayImage);
    }
```

> Setiap ada perubahan pada nama pemain atau interaksi lainnya, memanggil updateMessage() untuk memperbarui gambar tampilan.

**snake (Musuh yang harus dilawan pemain pada world “Level”)**

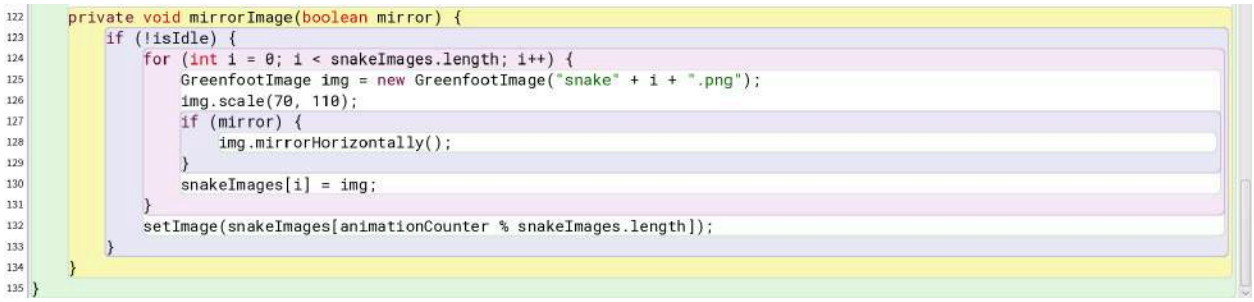


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class snake extends Actor {
    private GreenfootImage idleImage = new GreenfootImage("snake.png");
    private GreenfootImage[] snakeImages = new GreenfootImage[5];
    private int animationCounter = 0;
    private int frameDelay = 10;
    private int frameDelayCounter = 0;
    private boolean isAlive = true;
    private boolean isMirrored = false;
    private boolean isIdle = true;
    private int idleDelayTimer = 300; // 5 detik pada 60 FPS
    private int idleDelayCounter = 0;
    private boolean playerAlreadyHit = false; // player hit stat

    public snake() {
        idleImage.scale(70, 110);
        setImage(idleImage);
        for (int i = 0; i < snakeImages.length; i++) {
            snakeImages[i] = new GreenfootImage("snake" + i + ".png");
            snakeImages[i].scale(70, 110);
        }
    }
```



```
    public void act() {
        if (isAlive) {
            if (isIdle) {
                if (idleDelayCounter >= idleDelayTimer) {
                    isIdle = false;
                    idleDelayCounter = 0;
                    animationCounter = 0;
                } else {
                    idleDelayCounter++;
                }
            } else {
                animateSnake();
                if (animationCounter >= snakeImages.length) {
                    isIdle = true;
                    setImage(idleImage);
                }
            }

            checkForHitByGatotAttackEffect();
            if (getWorld() != null) {
                checkForHitByPlayer();
            }

            if (getWorld() != null) {
                checkAndMirrorImage();
            }
        }
```

```
    }

    private void animateSnake() {
        if (frameDelayCounter++ % frameDelay == 0) {
            Gatot_StandBy player = (Gatot_StandBy) getWorld().getObjects(Gatot_StandBy.class).get(0);
            if (player != null) {
                int snakeX = getX();
                int playerX = player.getX();
                if (snakeX > playerX) {
                    setImage(getMirroredImage(snakeImages[animationCounter % snakeImages.length]));
                } else {
                    setImage(snakeImages[animationCounter % snakeImages.length]);
                }
            }
            animationCounter++;
        }
    }

    private GreenfootImage getMirroredImage(GreenfootImage originalImage) {
        GreenfootImage mirroredImage = new GreenfootImage(originalImage);
        mirroredImage.mirrorHorizontally();
        mirroredImage.scale(70, 110);
        return mirroredImage;
    }

    private void checkForHitByGatotAttackEffect() {
        // snake hit klo bkn posisi idle
        if (!isIdle && isTouching(GatotAttackEffect.class)) {
            isAlive = false;
            removeTouching(GatotAttackEffect.class);

            // +skor
            BaseWorld world = (BaseWorld) getWorld();
            world.getCounter().addScore(1);

            getWorld().removeObject(this);
            return;
        }
    }

    private void checkForHitByPlayer() {
        Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);

        if (player != null && !isIdle) {
            if (!playerAlreadyHit) {
                playerAlreadyHit = true;
                BaseWorld world = (BaseWorld) getWorld();
                world.getHealthBar().decreaseLife(); // -nyawa
            }
        } else {
            playerAlreadyHit = false; // reset status jika tidak menyentuh lagi
        }
    }

    private void checkAndMirrorImage() {
        Gatot_StandBy player = (Gatot_StandBy) getWorld().getObjects(Gatot_StandBy.class).get(0);

        if (player != null) {
            int snakeX = getX();
            int playerX = player.getX();
            if (snakeX > playerX && !isMirrored) {
                mirrorImage(true);
                isMirrored = true;
            } else if (snakeX < playerX && isMirrored) {
                mirrorImage(false);
                isMirrored = false;
            }
        }
    }
```



```
122     private void mirrorImage(boolean mirror) {
123         if (!isIdle) {
124             for (int i = 0; i < snakeImages.length; i++) {
125                 GreenfootImage img = new GreenfootImage("snake" + i + ".png");
126                 img.scale(70, 110);
127                 if (mirror) {
128                     img.mirrorHorizontally();
129                 }
130                 snakeImages[i] = img;
131             }
132             setImage(snakeImages[animationCounter % snakeImages.length]);
133         }
134     }
135 }
```

Penjelasan :

```
private GreenfootImage idleImage = new GreenfootImage("snake.png");
private GreenfootImage[] snakeImages = new GreenfootImage[5];
private int animationCounter = 0;
private int frameDelay = 10;
private int frameDelayCounter = 0;
private boolean isAlive = true;
private boolean isMirrored = false;
private boolean isIdle = true;
private int idleDelayTimer = 300; // 5 detik pada 60 FPS
private int idleDelayCounter = 0;
private boolean playerAlreadyHit = false; // player hit stat
```

> idleImage: Gambar saat ular dalam keadaan idle.
> snakeImages: Array gambar animasi ular saat bergerak.
> animationCounter: Menghitung frame animasi yang sedang ditampilkan.
> frameDelay & frameDelayCounter: Mengatur kecepatan pergantian frame animasi.
> isAlive: Status hidup/mati ular.
> isMirrored: Status apakah gambar ular dicerminkan.
> isIdle: Status apakah ular sedang idle.
> idleDelayTimer & idleDelayCounter: Timer untuk mengatur durasi idle sebelum ular mulai bergerak.
> playerAlreadyHit: Mencegah ular menyerang pemain secara berulang saat bertabrakan.

```
public snake() {
    idleImage.scale(70, 110);
    setImage(idleImage);
    for (int i = 0; i < snakeImages.length; i++) {
      snakeImages[i] = new GreenfootImage("snake" + i + ".png");
      snakeImages[i].scale(70, 110);
    }
  }
```

> Membuat gambar ular dalam keadaan idle (idleImage) dan animasi ular (snakeImages). Gambar-gambar ini diubah ukurannya menjadi 70x110 pixel, lalu gambar idle diatur sebagai tampilan awal.

```
public void act() {
    if (isAlive) {
        if (isIdle) {
            if (idleDelayCounter >= idleDelayTimer) {
                isIdle = false;
                idleDelayCounter = 0;
                animationCounter = 0;
            } else {
                idleDelayCounter++;
            }
        } else {
            animateSnake();
            if (animationCounter >= snakeImages.length) {
                isIdle = true;
                setImage(idleImage);
            }
        }

        checkForHitByGatotAttackEffect();
        if (getWorld() != null) {
            checkForHitByPlayer();
        }

        if (getWorld() != null) {
            checkAndMirrorImage();
        }
    }
}
```

> Jika ular hidup (isAlive), ia akan berada dalam keadaan idle hingga timer selesai. Setelah itu, ular akan menjalankan animasi dan kembali idle setelah selesai. Selain itu, dicek apakah ular tersentuh efek serangan atau pemain, dan diatur pencerminan gambar berdasarkan posisi relatif ular terhadap pemain.

```
private void animateSnake() {
    if (frameDelayCounter++ % frameDelay == 0) {
```

```
                                        Gatot_StandBy player = (Gatot_StandBy)
getWorld().getObjects(Gatot_StandBy.class).get(0);
        if (player != null) {
            int snakeX = getX();
            int playerX = player.getX();
            if (snakeX > playerX) {
                                    setImage(getMirroredImage(snakeImages[animationCounter %
snakeImages.length]));
            } else {
                setImage(snakeImages[animationCounter % snakeImages.length]);
            }
        }
        animationCounter++;
    }
  }
```

> Mengatur animasi ular dengan mengganti gambar berdasarkan frame saat ini. Jika posisi ular di kanan pemain, gambar dicerminkan sebelum ditampilkan. Perubahan gambar dilakukan dengan interval tertentu berdasarkan frameDelay.

```
private GreenfootImage getMirroredImage(GreenfootImage originalImage) {
    GreenfootImage mirroredImage = new GreenfootImage(originalImage);
    mirroredImage.mirrorHorizontally();
    mirroredImage.scale(70, 110);
    return mirroredImage;
  }
```

> Menciptakan salinan gambar yang dicerminkan secara horizontal menggunakan mirrorHorizontally(). Gambar dicerminkan hanya saat ular berada di kanan pemain, lalu diatur ulang skalanya.

```
private void checkForHitByGatotAttackEffect() {
    // snake hit klo bkn posisi idle
    if (!isIdle && isTouching(GatotAttackEffect.class)) {
        isAlive = false;
        removeTouching(GatotAttackEffect.class);

        // +skor
        BaseWorld world = (BaseWorld) getWorld();
        world.getCounter().addScore(1);

        getWorld().removeObject(this);
```

```
            return;
        }
    }
```
> Memeriksa apakah ular tersentuh oleh objek GatotAttackEffect saat tidak idle. Jika iya, ular dihapus dari dunia, skor pemain ditambah 1, dan efek serangan dihapus.

```
private void checkForHitByPlayer() {
        Gatot_StandBy player = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);

        if (player != null && !isIdle) {
            if (!playerAlreadyHit) {
                playerAlreadyHit = true;
                BaseWorld world = (BaseWorld) getWorld();
                world.getHealthBar().decreaseLife(); // -nyawa
            }
        } else {
            playerAlreadyHit = false; // reset status jika tidak menyentuh lagi
        }
    }
```
> Memeriksa apakah ular bertabrakan dengan pemain (Gatot_StandBy) saat tidak idle. Jika iya dan ular belum memukul sebelumnya, status serangan ular diubah (playerAlreadyHit), dan nyawa pemain berkurang. Status serangan direset jika tabrakan selesai.

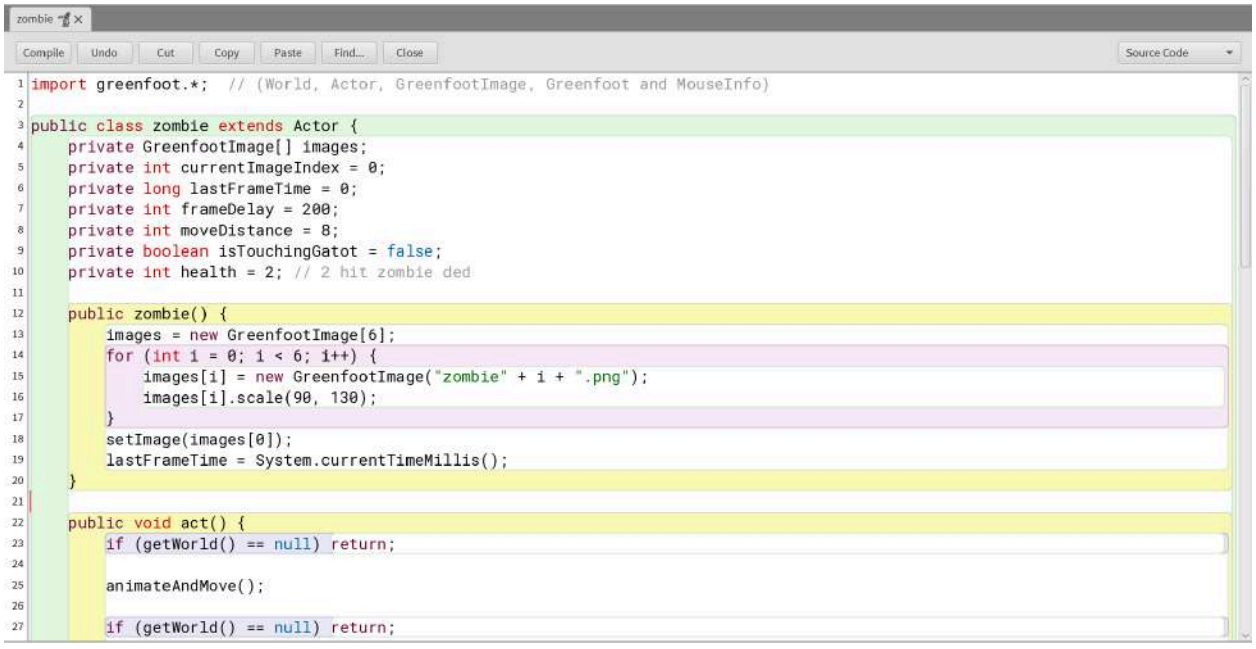
```
private void checkAndMirrorImage() {
                                                Gatot_StandBy player = (Gatot_StandBy) getWorld().getObjects(Gatot_StandBy.class).get(0);

        if (player != null) {
            int snakeX = getX();
            int playerX = player.getX();
            if (snakeX > playerX && !isMirrored) {
                mirrorImage(true);
                isMirrored = true;
            } else if (snakeX < playerX && isMirrored) {
                mirrorImage(false);
                isMirrored = false;
            }
        }
    }
```

> Mengatur pencerminan gambar ular berdasarkan posisi relatif dengan pemain. Jika ular di kanan pemain, gambar dicerminkan; jika di kiri, pencerminan dibalik.

```
private void mirrorImage(boolean mirror) {
      if (!isIdle) {
         for (int i = 0; i < snakeImages.length; i++) {
            GreenfootImage img = new GreenfootImage("snake" + i + ".png");
            img.scale(70, 110);
            if (mirror) {
               img.mirrorHorizontally();
            }
            snakeImages[i] = img;
         }
         setImage(snakeImages[animationCounter % snakeImages.length]);
      }
   }
```

> Mengubah semua gambar animasi ular menjadi versi cerminnya jika dibutuhkan. Gambar diperbarui sesuai status pencerminan (isMirrored) untuk memastikan konsistensi animasi.

**zombie (Musuh yang harus dilawan pemain pada world “Level”)**



```
        checkHitByAttack();

        if (getWorld() == null) return;
        checkHitByGatot();
    }

    private void animateAndMove() {
        if (System.currentTimeMillis() - lastFrameTime >= frameDelay) {
            currentImageIndex = (currentImageIndex + 1) % images.length;
            setImage(images[currentImageIndex]);
            setLocation(getX() - moveDistance, getY());
            lastFrameTime = System.currentTimeMillis();
        }
    }

    private void checkHitByAttack() {
        if (getWorld() == null) return;
        GatotAttackEffect attack = (GatotAttackEffect) getOneIntersectingObject(GatotAttackEffect.class);

        if (attack != null) {
            BaseWorld world = (BaseWorld) getWorld();

            if (world != null) {
                world.getCounter().addScore(1); // +skor
                world.removeObject(attack); // hapus serangan

                health--; // kurangi nyawa zombie
```

```
                if (health <= 0) {
                    world.getCounter().addScore(1); // +Skor
                    world.removeObject(this); //  zombie hilang
                }
            }
        }
    }

    private void checkHitByGatot() {
        if (getWorld() == null) return;
        Gatot_StandBy gatot = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);

        if (gatot != null && !isTouchingGatot) {
            isTouchingGatot = true; // zombie touch gatot
            BaseWorld world = (BaseWorld) getWorld();

            if (world != null) { // dunia masih ada sebelum mengakses HealthBar
                HealthBar healthBar = world.getHealthBar();
                healthBar.decreaseLife(); // -nyawa
            }
        } else if (gatot == null) {
            isTouchingGatot = false; // reset
        }
    }
}
```

Penjelasan :

```
    private GreenfootImage[] images;
    private int currentImageIndex = 0;
    private long lastFrameTime = 0;
    private int frameDelay = 200;
    private int moveDistance = 8;
    private boolean isTouchingGatot = false;
    private int health = 2; // 2 hit zombie ded
```

> images: Array untuk menyimpan gambar animasi zombie.
> currentImageIndex: Indeks gambar yang sedang ditampilkan untuk animasi.
> lastFrameTime: Menyimpan waktu terakhir frame animasi berubah.
> frameDelay: Durasi jeda antar frame animasi dalam milidetik.
> moveDistance: Jarak perpindahan zombie setiap frame.
> isTouchingGatot: Status apakah zombie sedang menyentuh Gatot.

> health: Jumlah nyawa zombie, berkurang setiap kali terkena serangan hingga hilang.

```
public zombie() {
        images = new GreenfootImage[6];
        for (int i = 0; i < 6; i++) {
            images[i] = new GreenfootImage("zombie" + i + ".png");
            images[i].scale(90, 130);
        }
        setImage(images[0]);
        lastFrameTime = System.currentTimeMillis();
    }
```

> Array gambar animasi zombie dan mengatur ukuran gambar.
> Gambar awal diatur ke gambar pertama dalam array (images[0]).
> Menyimpan waktu saat ini ke lastFrameTime untuk animasi.

```
public void act() {
        if (getWorld() == null) return;

        animateAndMove();

        if (getWorld() == null) return;
        checkHitByAttack();

        if (getWorld() == null) return;
        checkHitByGatot();
    }
```

> Memastikan objek masih ada di dunia sebelum melanjutkan eksekusi.
> Menjalankan tiga fungsi utama secara berurutan:

- animateAndMove() untuk animasi dan pergerakan zombie.
- checkHitByAttack() untuk mendeteksi serangan oleh Gatot.
- checkHitByGatot() untuk mendeteksi tabrakan langsung dengan Gatot.

```
private void animateAndMove() {
        if (System.currentTimeMillis() - lastFrameTime >= frameDelay) {
            currentImageIndex = (currentImageIndex + 1) % images.length;
            setImage(images[currentImageIndex]);
            setLocation(getX() - moveDistance, getY());
            lastFrameTime = System.currentTimeMillis();
        }
    }
```

> Mengganti frame animasi berdasarkan waktu dengan mengecek frameDelay.
> Setelah mengubah frame, zombie bergerak ke kiri sejauh moveDistance.

```
private void checkHitByAttack() {
    if (getWorld() == null) return;
                                        GatotAttackEffect attack = (GatotAttackEffect)
getOneIntersectingObject(GatotAttackEffect.class);

    if (attack != null) {
      BaseWorld world = (BaseWorld) getWorld();

      if (world != null) {
        world.getCounter().addScore(1); // +skor
        world.removeObject(attack); // hapus serangan

        health--; // kurangi nyawa zombie
        if (health <= 0) {
          world.getCounter().addScore(1); // +Skor
          world.removeObject(this); //  zombie hilang
        }
      }
    }
  }
```

> Mengecek apakah zombie terkena GatotAttackEffect. Jika kena, maka menambah skor dunia, menghapus objek serangan, mengurangi nyawa zombie (health). Jika nyawa habis, zombie dihapus dari dunia dan skor bertambah lagi.

```
private void checkHitByGatot() {
    if (getWorld() == null) return;
    Gatot_StandBy gatot = (Gatot_StandBy) getOneIntersectingObject(Gatot_StandBy.class);

    if (gatot != null && !isTouchingGatot) {
      isTouchingGatot = true; // zombie touch gatot
      BaseWorld world = (BaseWorld) getWorld();

      if (world != null) { // dunia masih ada sebelum mengakses HealthBar
        HealthBar healthBar = world.getHealthBar();
        healthBar.decreaseLife(); // -nyawa
      }
    } else if (gatot == null) {
```

```
            isTouchingGatot = false; // reset
        }
    }
```

> Jika menyentuh Gatot_StandBy untuk pertama kali (!isTouchingGatot), zombie mengurangi nyawa pemain melalui HealthBar.
> Jika tidak menyentuh, status isTouchingGatot direset untuk memungkinkan serangan berikutnya.

**Gambar pertama untuk soal pada world "FiLLiT" terdapat fungsi yang memungkinkan perubahan gambar objek secara dinamis selama permainan.**
> setImage(imageName): Mengganti gambar objek dengan gambar baru sesuai nama file yang diberikan melalui parameter imageName.

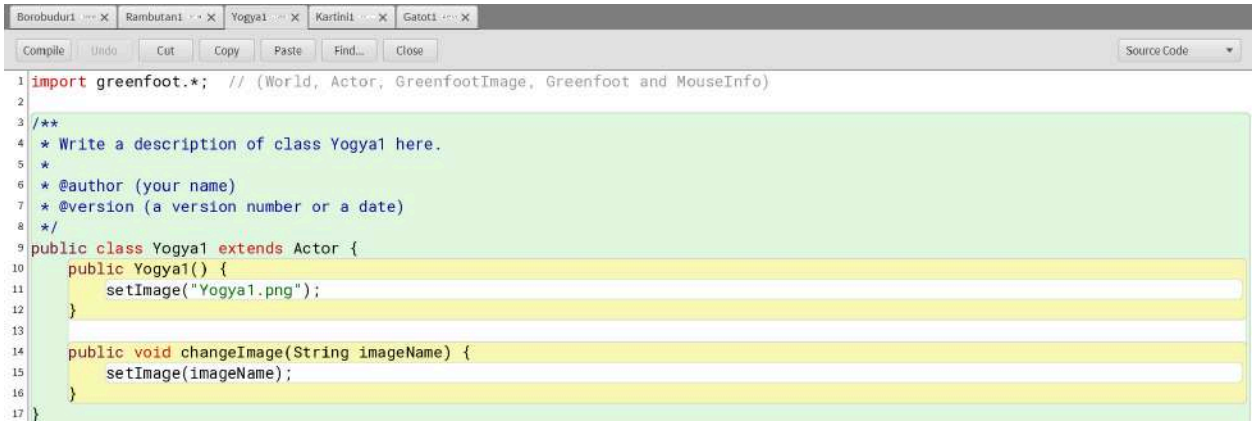
**Borobudur1**


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Borobudur1 extends Actor {
    public Borobudur1() {
        setImage("Borobudur1.png");
    }

    public void changeImage(String imageName) {
        setImage(imageName);
    }
}
```

**Rambutan1**


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Yogya1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Rambutan1 extends Actor {
    public Rambutan1() {
        setImage("Rambutan1.png");
    }

    public void changeImage(String imageName) {
        setImage(imageName);
    }
}
```

## Yogya1

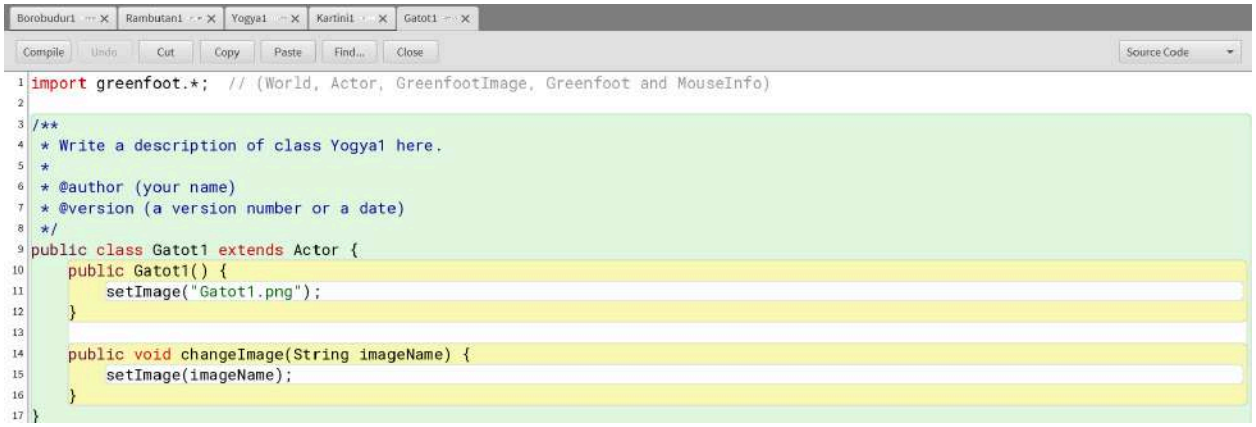



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Yogya1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Yogya1 extends Actor {
    public Yogya1() {
        setImage("Yogya1.png");
    }

    public void changeImage(String imageName) {
        setImage(imageName);
    }
}
```

## Kartini1


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Yogya1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Kartini1 extends Actor {
    public Kartini1() {
        setImage("Kartini1.png");
    }

    public void changeImage(String imageName) {
        setImage(imageName);
    }
}
```

## Gatot1




```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Yogya1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Gatot1 extends Actor {
    public Gatot1() {
        setImage("Gatot1.png");
    }

    public void changeImage(String imageName) {
        setImage(imageName);
    }
}
```

**NOTE :**
**Terdapat beberapa objek yang memilki kode berikut :**

```
public Nama_Kelas(int height, int width)
   {
   getImage().scale(height, width);
   }
```

**Fungsi : Mengatur ukuran gambar (Image) dari objek kelas Nama_Kelas.**

**Buttons :**
**Objek buttons merupakan objek yang berfungsi sebagai tombol dimana apabila objek tersebut menerima input mouse klik dari pemain, maka objek akan menghentikan musik yang ada di world saat itu kemudian langsung melakukan transisi ke world yang dituju button tersebut.**

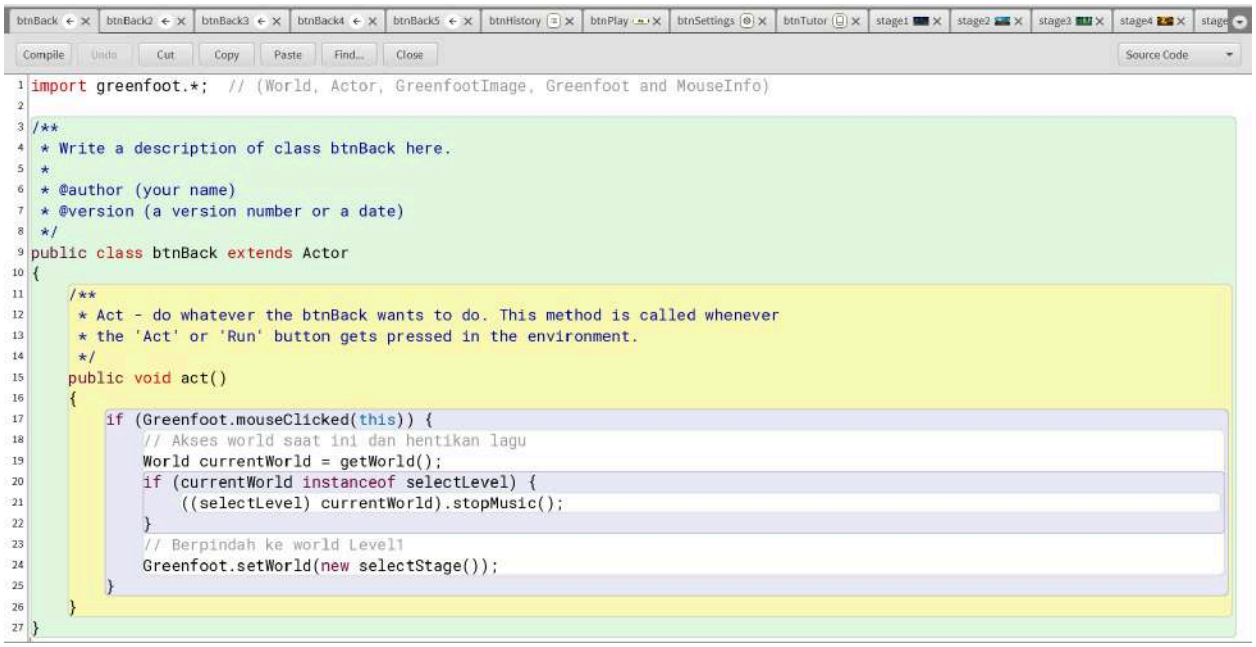
**btnAbout**

btnAbout | btnBack2 | btnBack3 | btnBack4 | btnBack5 | btnHistory | btnPlay | btnSettings | btnTutor | stage1 | stage2 | stage3 | stage4

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class btnAbout extends Actor
{
    /**
     * Act - do whatever the btnAbout wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            Counter.setGlobalScore(0); // skor global reset jadi 0

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof MainMenu) {
                ((MainMenu) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new aboutMenu());
        }
    }
}
```

## btnBack

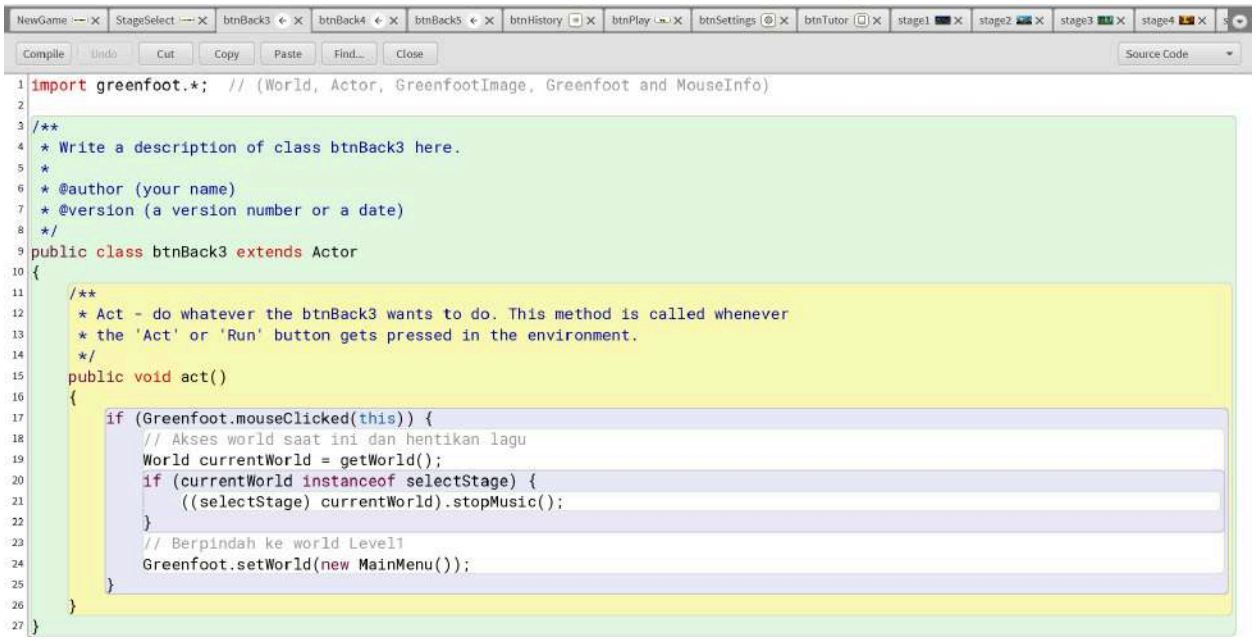


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class btnBack here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class btnBack extends Actor
{
    /**
     * Act - do whatever the btnBack wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectLevel) {
                ((selectLevel) currentWorld).stopMusic();
            }
            // Berpindah ke world Level1
            Greenfoot.setWorld(new selectStage());
        }
    }
}
```

## btnBack2

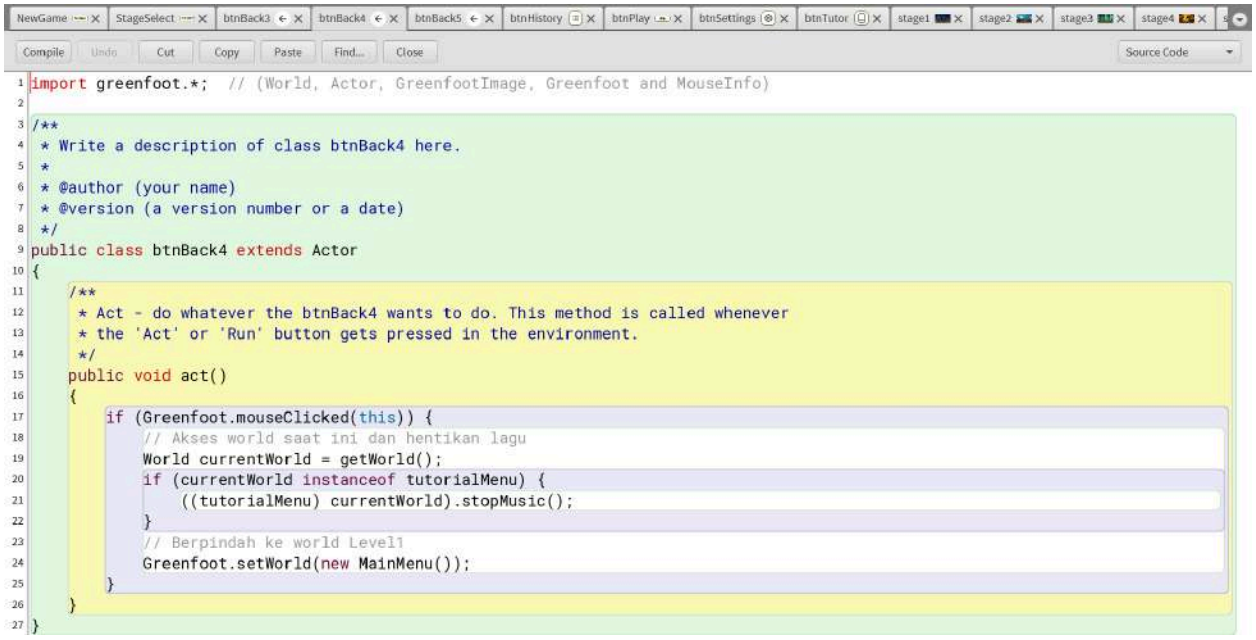
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class btnBack2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class btnBack2 extends Actor
{
    /**
     * Act - do whatever the btnBack2 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof historyMenu) {
                ((historyMenu) currentWorld).stopMusic();
            }
            // Berpindah ke world Level1
            Greenfoot.setWorld(new MainMenu());
        }
    }
}
```

## btnBack3



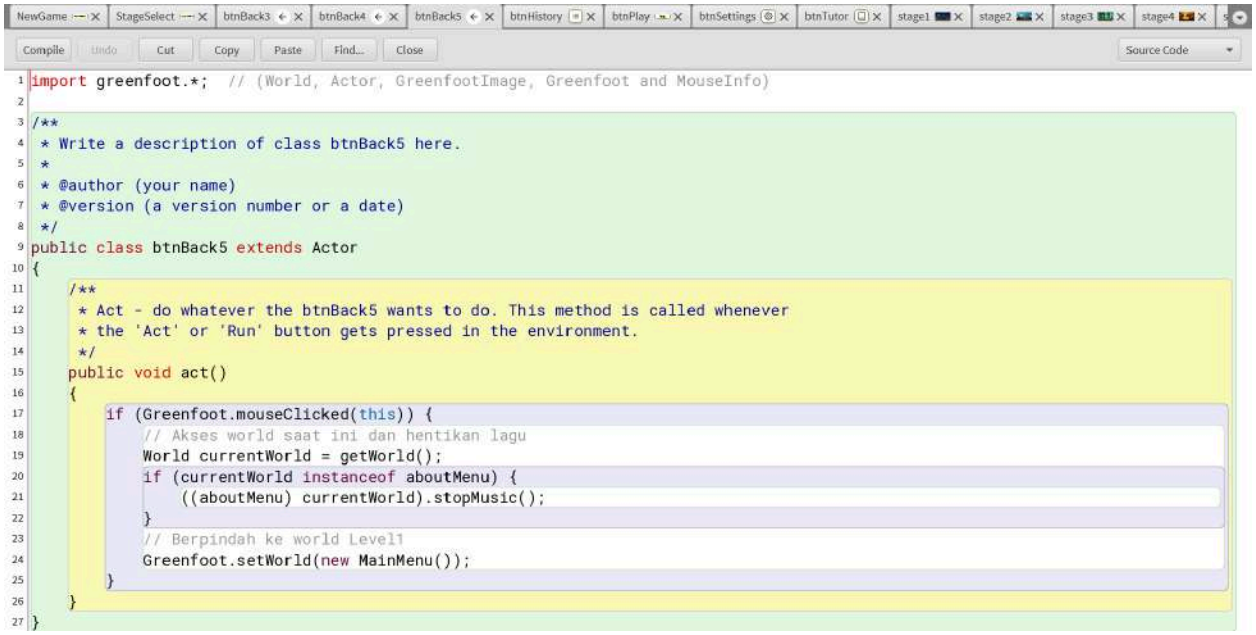
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class btnBack3 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class btnBack3 extends Actor
{
    /**
     * Act - do whatever the btnBack3 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectStage) {
                ((selectStage) currentWorld).stopMusic();
            }
            // Berpindah ke world Level1
            Greenfoot.setWorld(new MainMenu());
        }
    }
}
```

## btnBack4



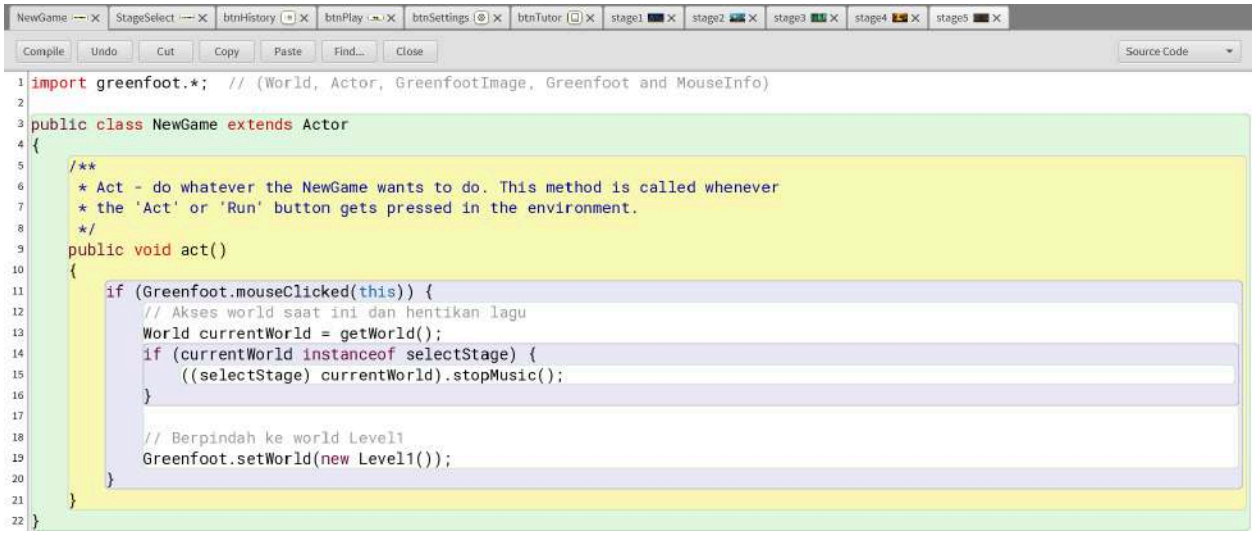
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class btnBack4 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class btnBack4 extends Actor
{
    /**
     * Act - do whatever the btnBack4 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof tutorialMenu) {
                ((tutorialMenu) currentWorld).stopMusic();
            }
            // Berpindah ke world Level1
            Greenfoot.setWorld(new MainMenu());
        }
    }
}
```

## btnBack5



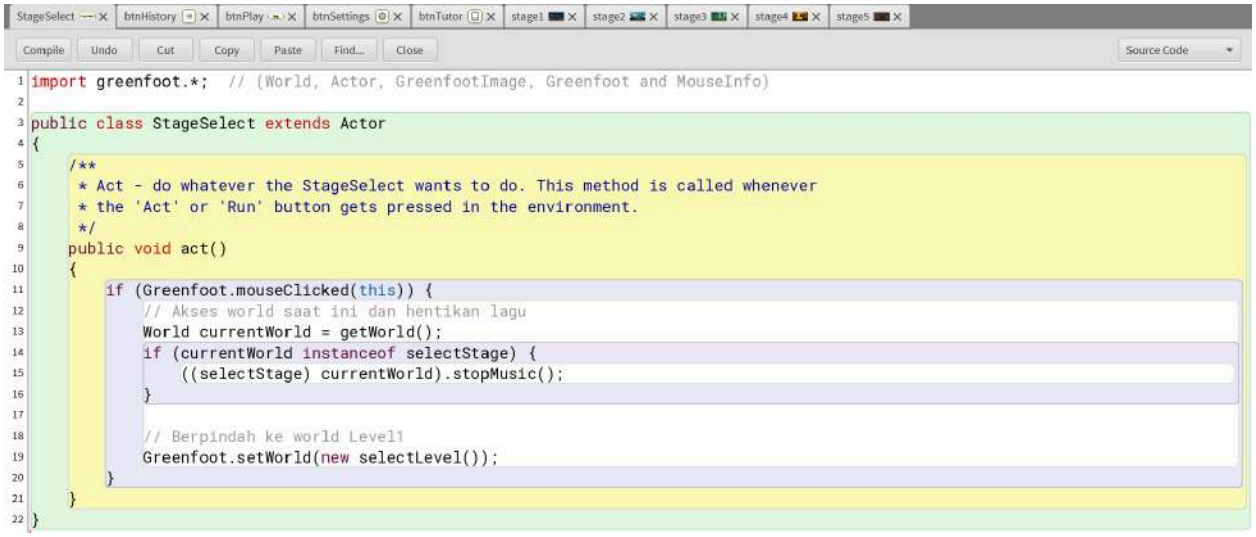
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class btnBack5 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class btnBack5 extends Actor
{
    /**
     * Act - do whatever the btnBack5 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof aboutMenu) {
                ((aboutMenu) currentWorld).stopMusic();
            }
            // Berpindah ke world Level1
            Greenfoot.setWorld(new MainMenu());
        }
    }
}
```

## NewGame



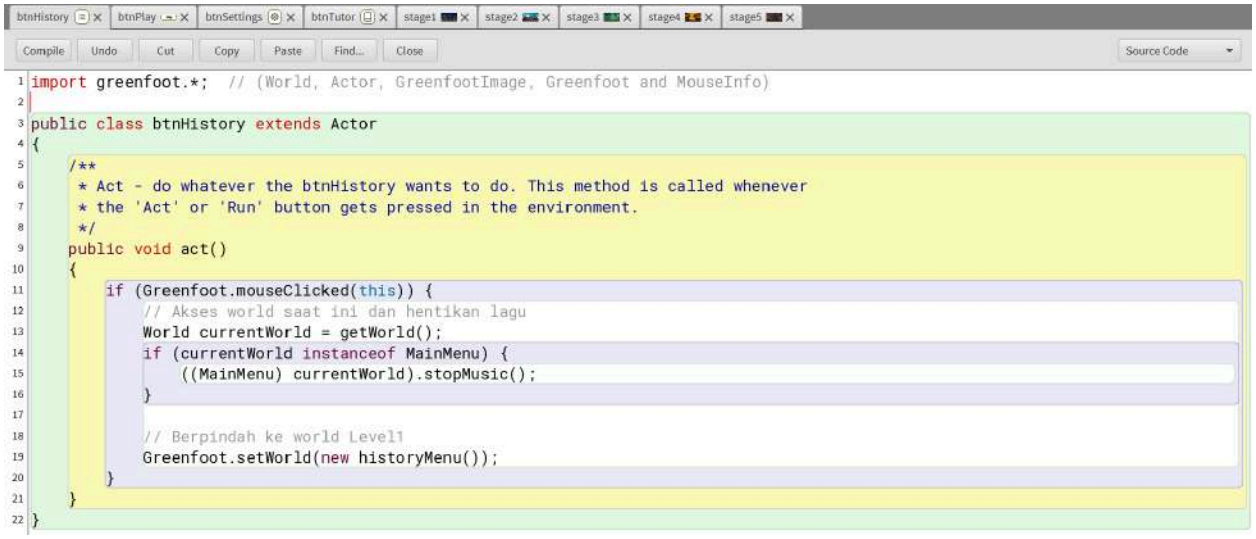
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class NewGame extends Actor
{
    /**
     * Act - do whatever the NewGame wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectStage) {
                ((selectStage) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new Level1());
        }
    }
}
```

## StageSelect



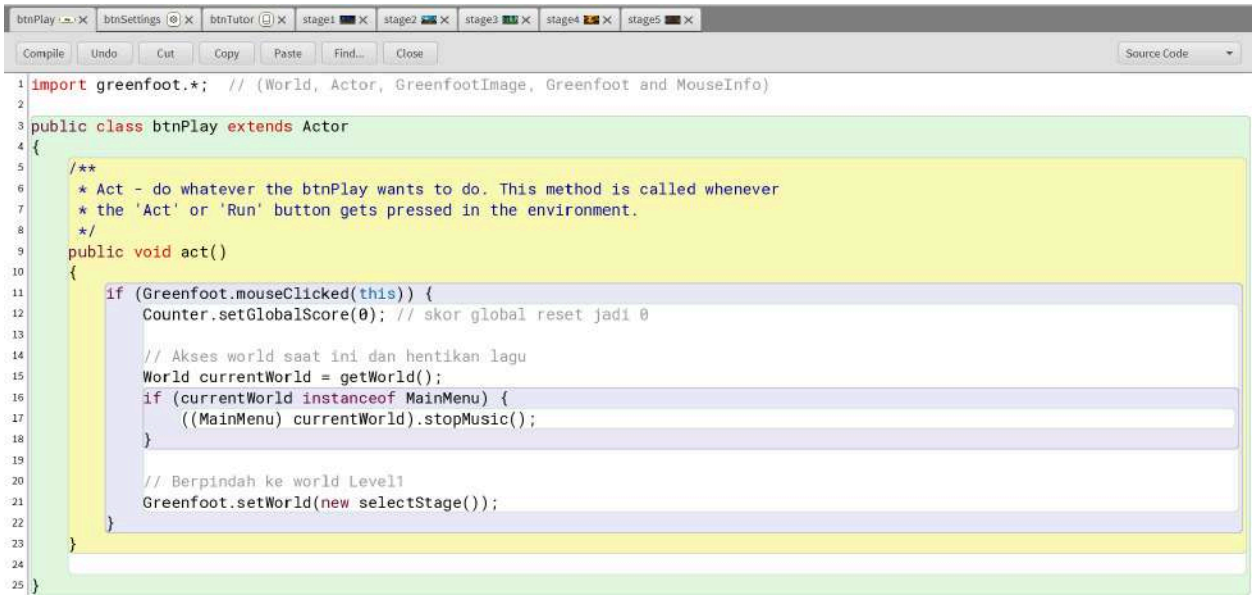
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class StageSelect extends Actor
{
    /**
     * Act - do whatever the StageSelect wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectStage) {
                ((selectStage) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new selectLevel());
        }
    }
}
```

## btnHistory



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class btnHistory extends Actor
{
    /**
     * Act - do whatever the btnHistory wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof MainMenu) {
                ((MainMenu) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new historyMenu());
        }
    }
}
```

## btnPlay

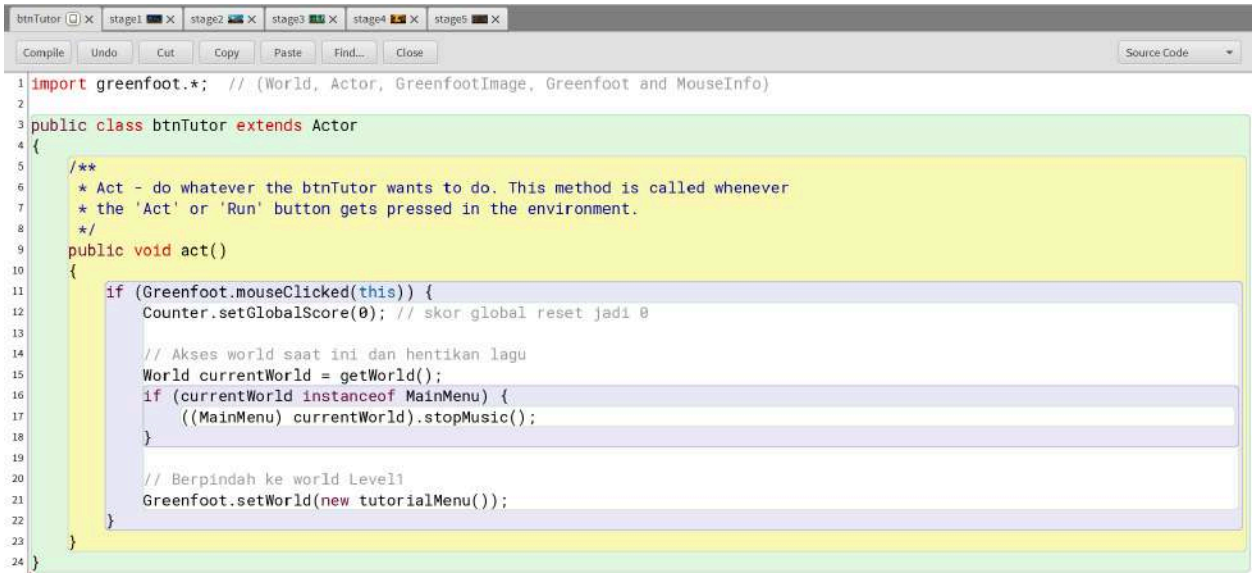



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class btnPlay extends Actor
{
    /**
     * Act - do whatever the btnPlay wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            Counter.setGlobalScore(0); // skor global reset jadi 0

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof MainMenu) {
                ((MainMenu) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new selectStage());
        }
    }

}
```

## btnSettings


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class btnSettings here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class btnSettings extends Actor
{
    /**
     * Act - do whatever the btnSettings wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

## btnTutor

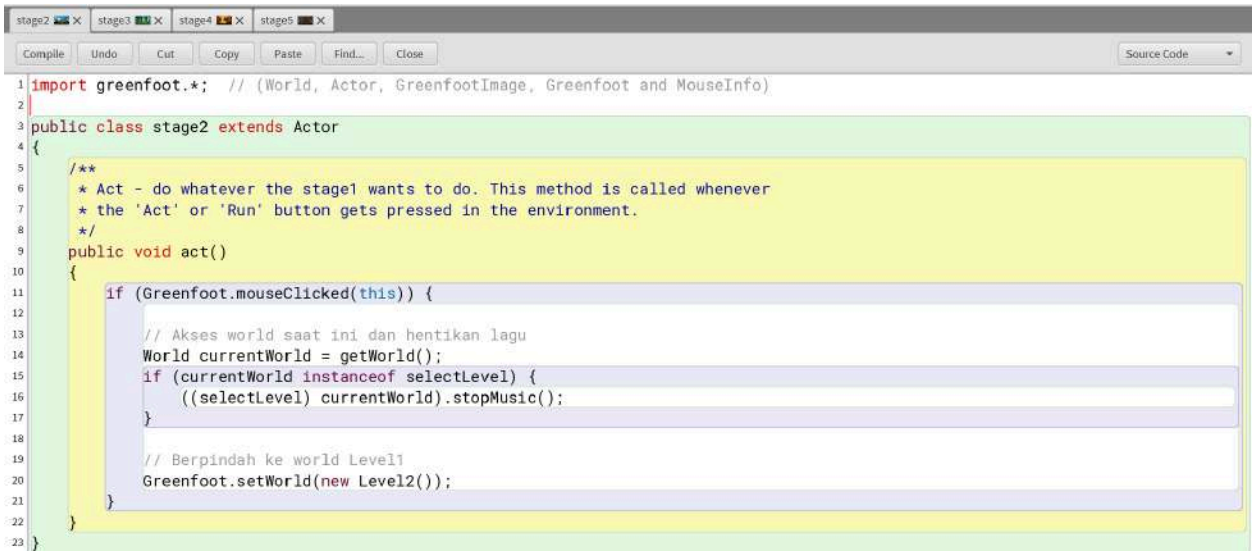



```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class btnTutor extends Actor
{
    /**
     * Act - do whatever the btnTutor wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {
            Counter.setGlobalScore(0); // skor global reset jadi 0

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof MainMenu) {
                ((MainMenu) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new tutorialMenu());
        }
    }
}
```

## stage1

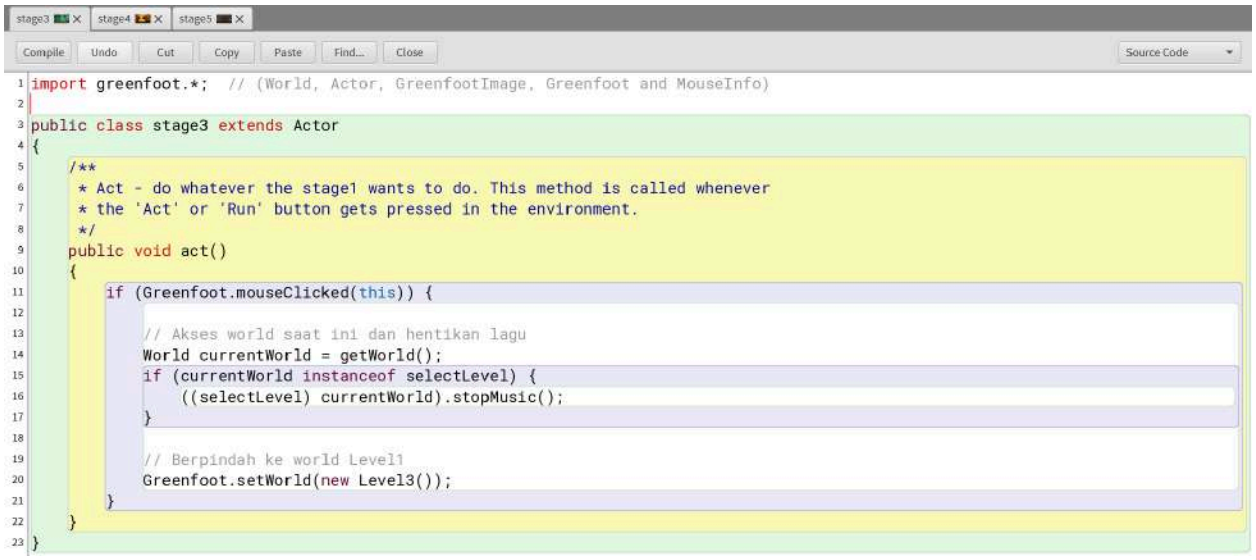

```java
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class stage1 extends Actor
{
    /**
     * Act - do whatever the stage1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectLevel) {
                ((selectLevel) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new Level1());
        }
    }
}
```

## stage2



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class stage2 extends Actor
{
    /**
     * Act - do whatever the stage1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectLevel) {
                ((selectLevel) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new Level2());
        }
    }
}
```

## stage3



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class stage3 extends Actor
{
    /**
     * Act - do whatever the stage1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectLevel) {
                ((selectLevel) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new Level3());
        }
    }
}
```

## stage4

stage4 stage5

Compile Undo Cut Copy Paste Find... Close Source Code

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class stage4 extends Actor
{
    /**
     * Act - do whatever the stage1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectLevel) {
                ((selectLevel) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new Level4());
        }
    }
}
```

## stage5

stage5

Compile Undo Cut Copy Paste Find... Close Source Code

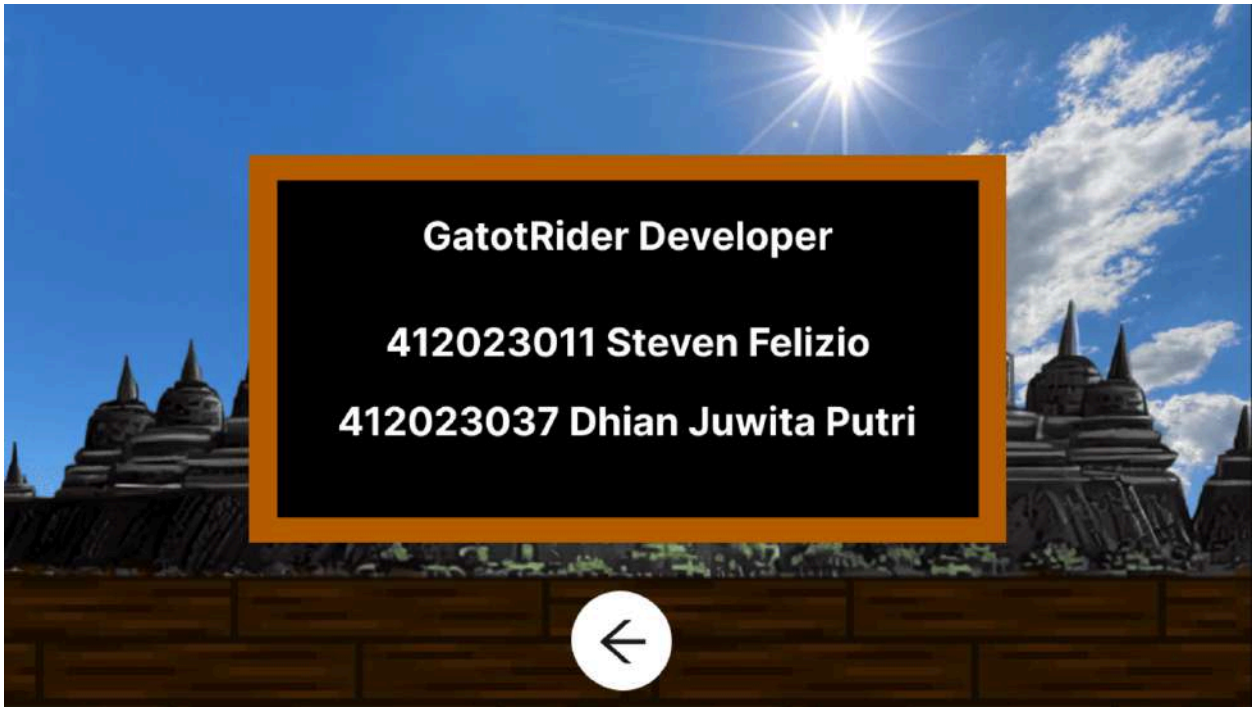
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class stage5 extends Actor
{
    /**
     * Act - do whatever the stage1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (Greenfoot.mouseClicked(this)) {

            // Akses world saat ini dan hentikan lagu
            World currentWorld = getWorld();
            if (currentWorld instanceof selectLevel) {
                ((selectLevel) currentWorld).stopMusic();
            }

            // Berpindah ke world Level1
            Greenfoot.setWorld(new Level5());
        }
    }
}
```

## PANDUAN BERMAIN *GAME GATOTRIDER*

***Main Menu***

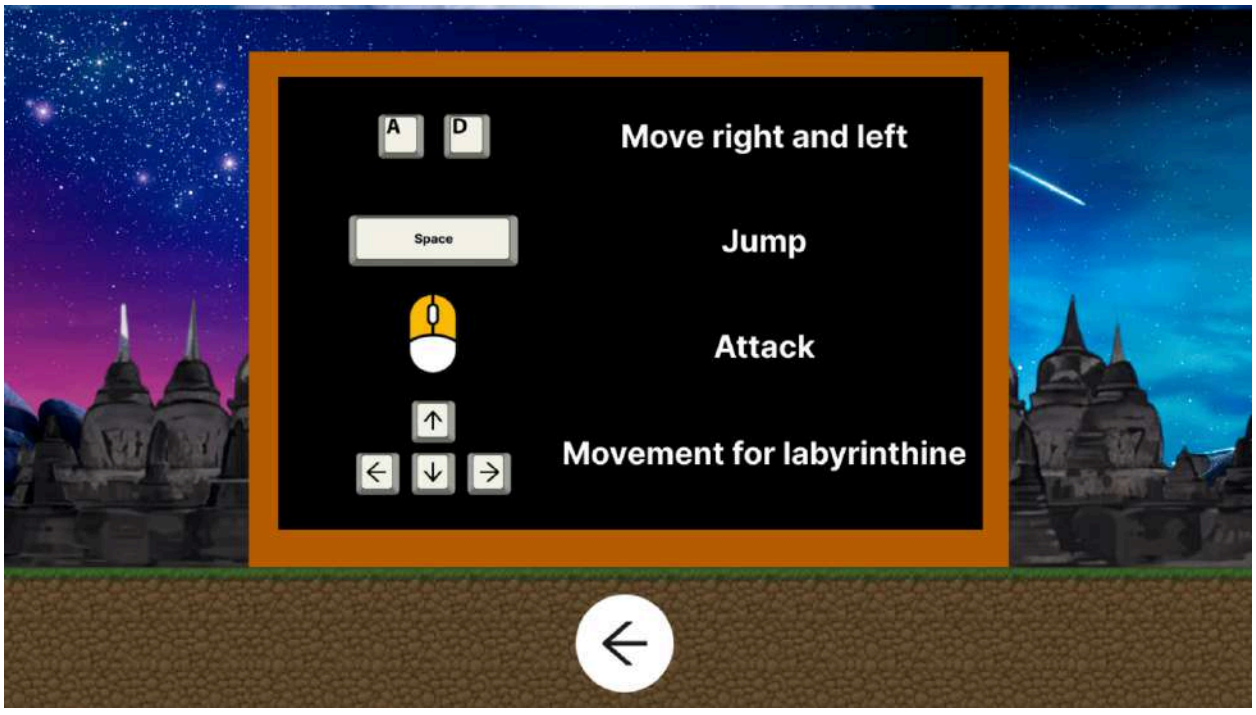

Saat pertama kali memasuki *game GatotRider*, pemain akan dibawa ke *main menu*. Di dalam *main menu* terdapat 4 tombol yaitu *Play, Scores, Tutorial,* dan *About*. Setiap tombol, apabila ditekan akan membawa player ke menu lain yang berbeda-beda. Di dalam menu *Scores*, terdapat *History scores* dari pemain yang telah memainkan *game*. Di dalam menu *Tutorial*, terdapat menu *Tutorial* yang berisi *Tutorial* singkat dari *game GatotRider*. Di dalam menu *About* akan berisi nama dari pengembang *game GatotRider*. Di dalam menu *Play,* terdapat dua pilihan untuk memulai *game* dari awal atau bisa memilih *stage* tertentu.

***About Menu***




Menu *About* berisi nama-nama dari pengembang *game GatotRider*.

***Tutorial Menu***



Menu *Tutorial* berisi tutorial singkat untuk *game GatotRider*.

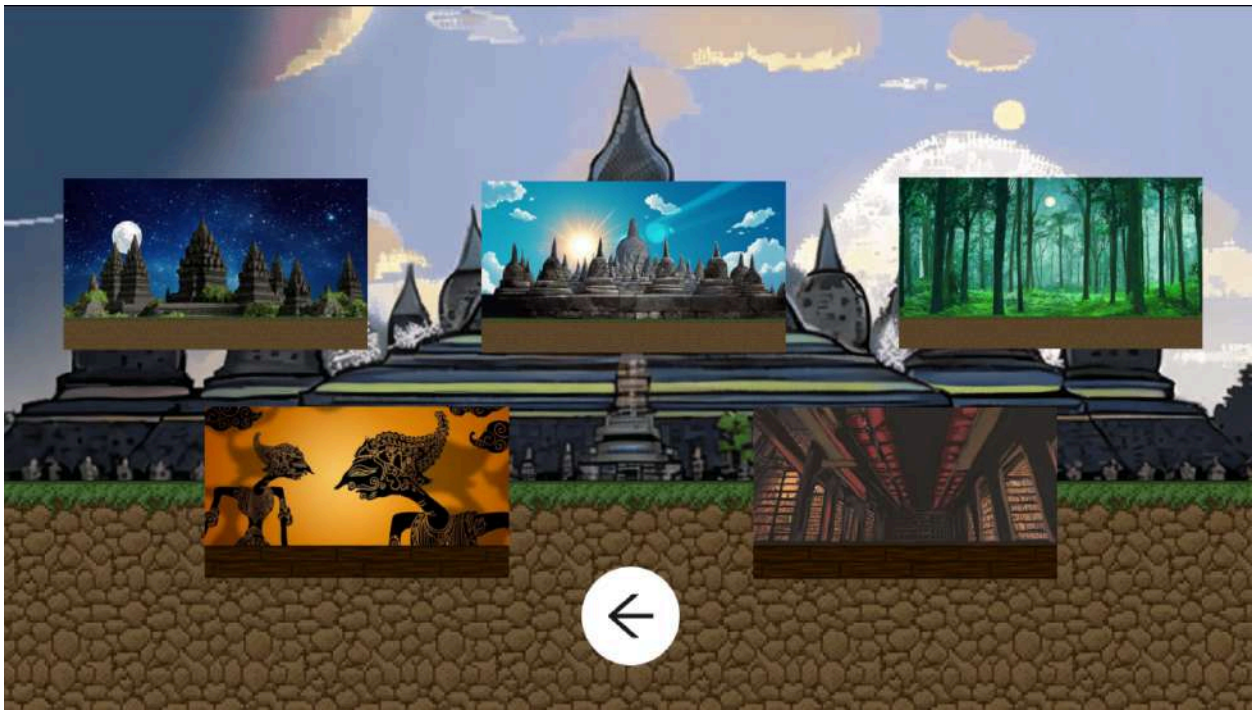
***Scores Menu***


Di dalam menu *Scores* terdapat *history scores* yang telah dicapai pemain ketika menyelesaikan *stage* dalam *game.* Terdapat juga *High Score*, yaitu *Score* tertinggi yang telah didapatkan pemain.

***Play Menu***


Terdapat 2 tombol dalam menu *Play*, yaitu *New Game* dan *Stage Select*. Apabila *New Game* ditekan, akan membawa pemain ke awal mula *game* atau *stage* 1. Sedangkan jika tombol *Stage Select* ditekan akan membawa player ke menu *Stage Select* untuk memilih 5 *stage* yang tersedia.

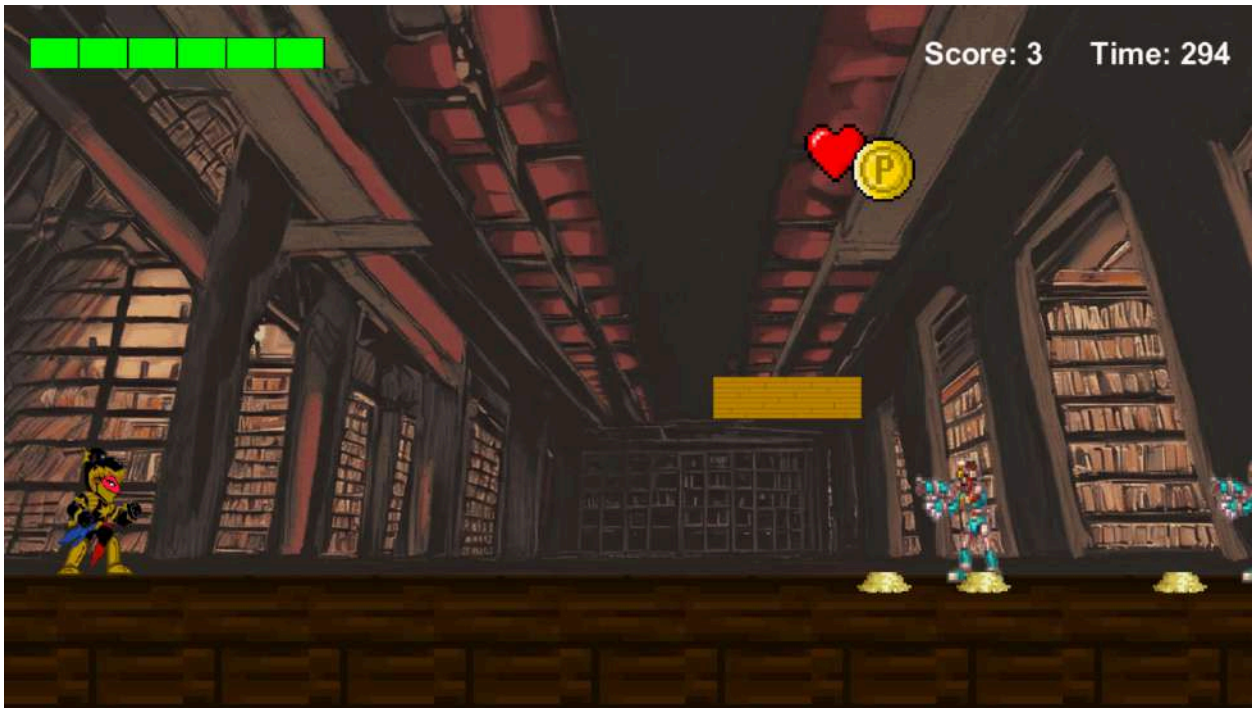
***Stage Select Menu***



Di dalam menu *Stage Select* terdapat 5 *stage* yang dapat dipilih untuk dimainkan. Setiap *stage* memiliki tingkat kesulitan yang berbeda-beda. Tingkat kesulitan setiap *stage* akan selalu

meningkat setiap *stage* berganti. Setiap *stage* juga memiliki latar, boss, dan ciri khas masing-masing sesuai dengan tema *game* yaitu Kekayaan Potensi Daerah Indonesia.

***Gameplay***

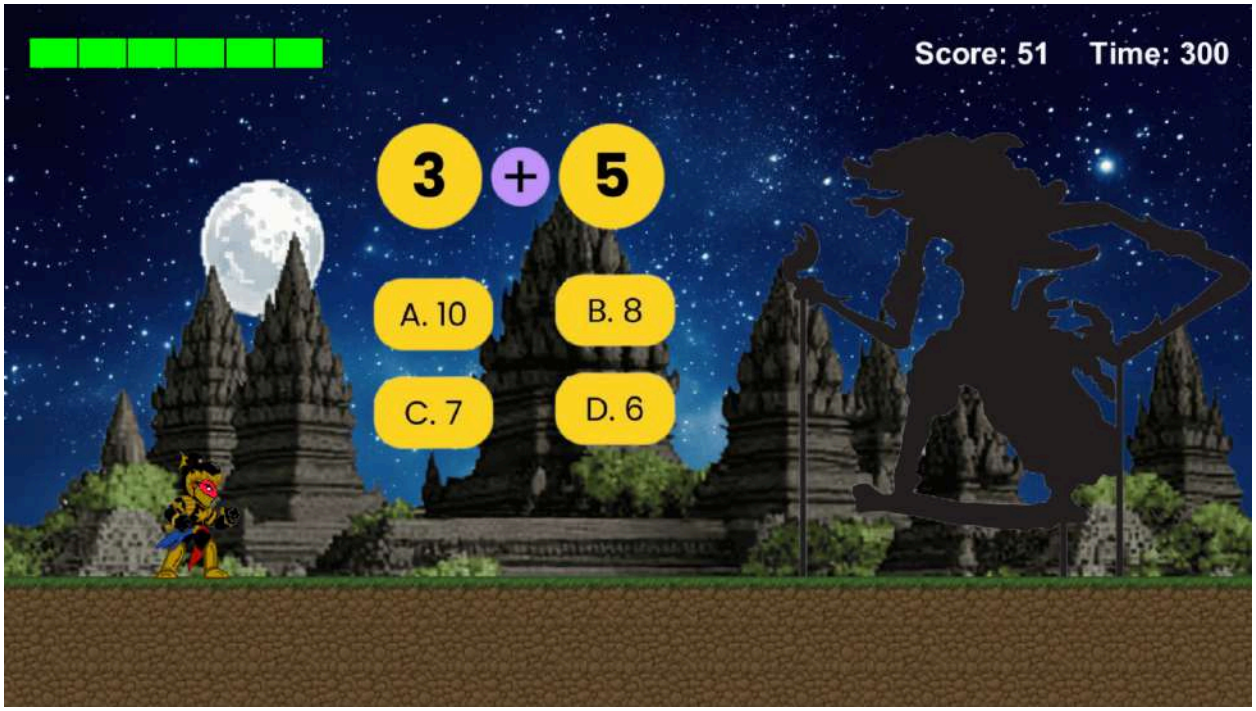


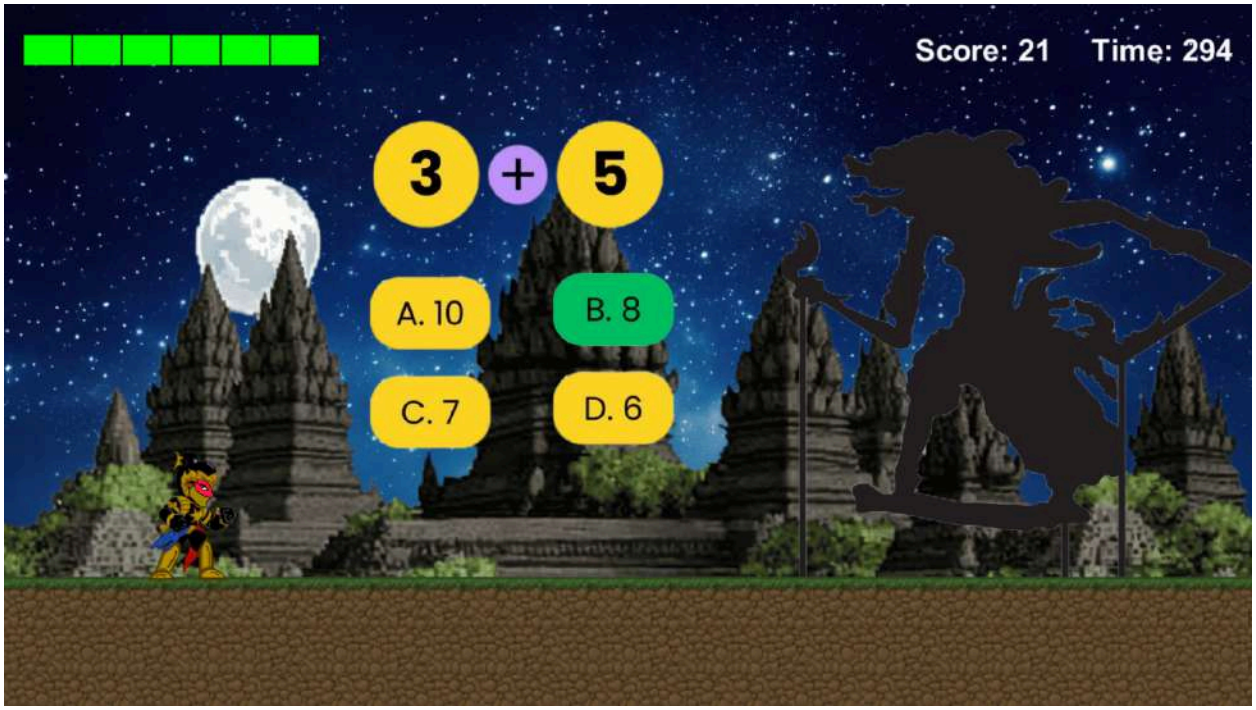

*Gameplay* dari *game GatotRider* adalah platformer dua dimensi di mana pemain akan melawan musuh-musuh yang ada, melakukan lompatan untuk menghindari musuh, mengambil koin *score* dan *health point*, dan mencapai titik *finish*. Untuk musuh *zombie*, apabila dikalahkan akan memberikan player 3 *score*, sedangkan untuk musuh *snake* akan memberikan player 1 poin. Koin yang bisa didapatkan oleh player akan memberikan 1 *score* untuk player.

***Fill It***


*Gameplay Fill It* akan mengajak player untuk mengenalkan kebudayaan Indonesia sekaligus pengenalan huruf. Pemain akan diberikan kuis untuk mengisi huruf yang kosong dari sebuah kata yang akan menjadi jawaban untuk menjawab pertanyaan wayang hitam. Cara menjawab pertanyaan cukup dengan mengisi huruf yang kosong dengan menggunakan *input* dari *keyboard*. Apabila pemain berhasil menjawab, maka akan dibawa untuk *mini game* yang selanjutnya, sedangkan apabila pemain gagal menjawab, maka pemain akan kehilangan satu nyawa.

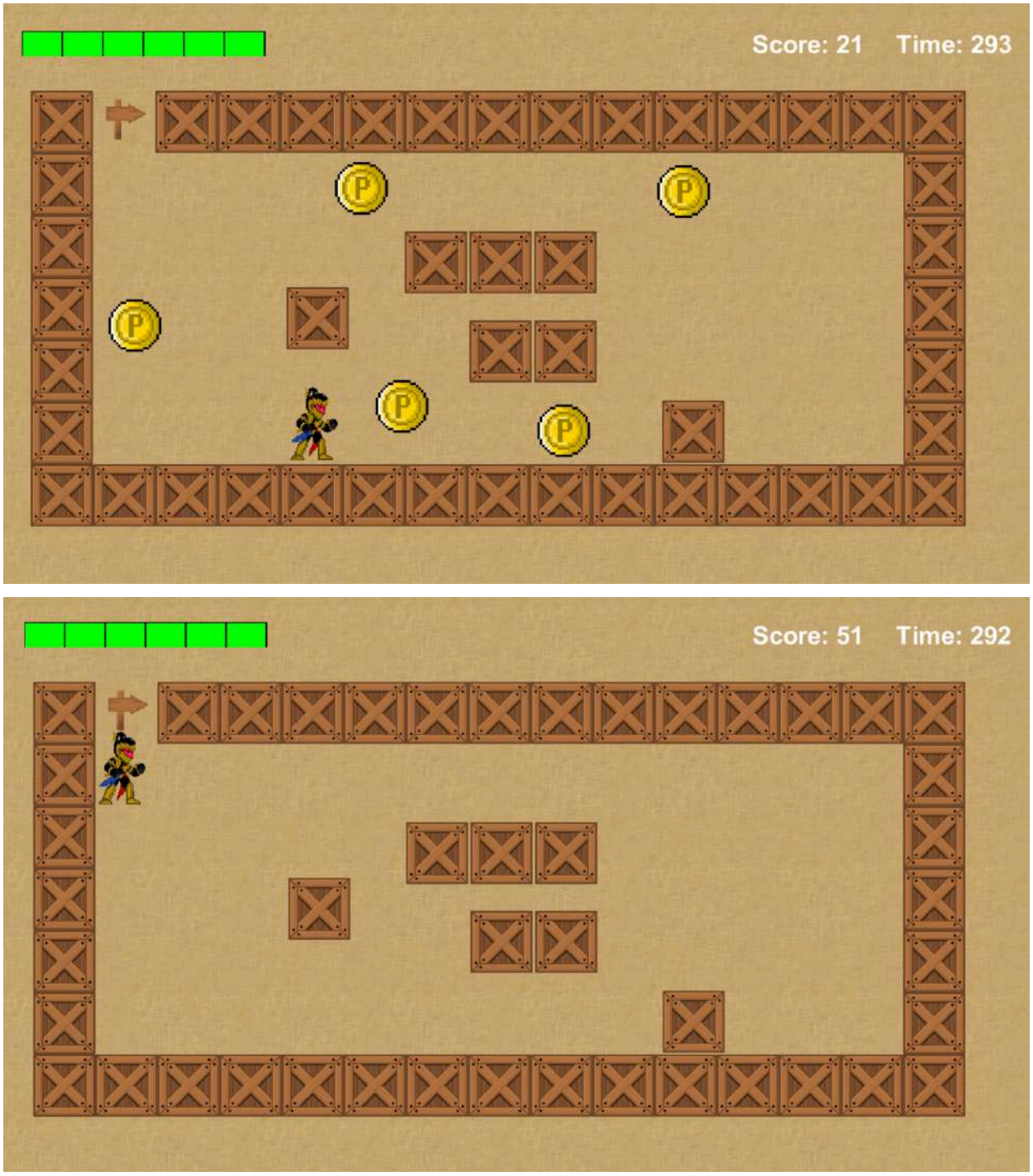
***NumQuest***






*Gameplay NumQuest* adalah *minigame* untuk melatih kemampuan berhitung (operasi penjumlahan dan pengurangan bilangan). Pemain akan diberikan kuis matematika oleh wayang hitam agar pemain bisa melanjutkan *progress* selanjutnya. Apabila pemain gagal menjawab kuis, maka nyawa player akan dikurangi satu.

***Labyrinthine***

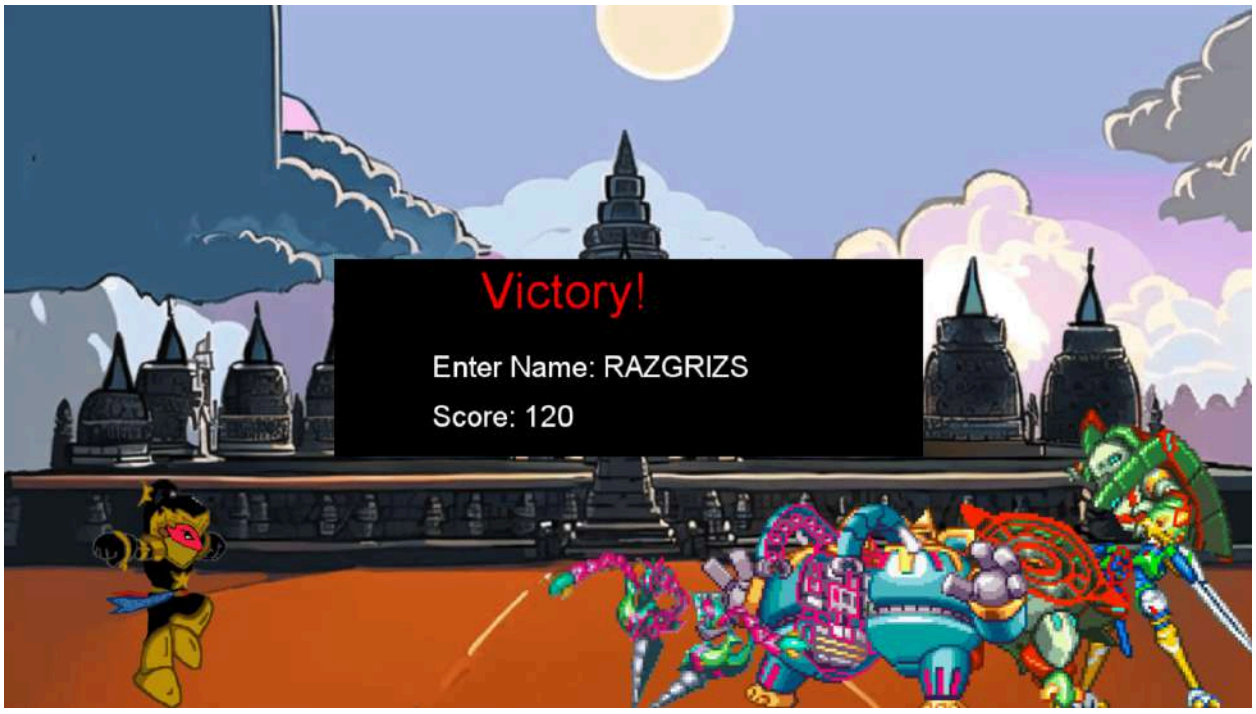



*Gameplay Labyrinthine* adalah *mini game* menyelesaikan labirin dengan koin *score* yang bisa diambil untuk menambah *score point*. Di dalam *mini game* ini, pemain akan menggunakan tombol *arrow* untuk mengarahkan *GatotRider* untuk mencapai titik *finish*. Apabila tombol *arrow* ditekan maka *GatotRider* akan terus berjalan sampai ujung, bertabrakan ke *crate*. Pemain harus berpikir bagaimana menyelesaikan labirin dengan ketentuan seperti itu.

***BossFight***

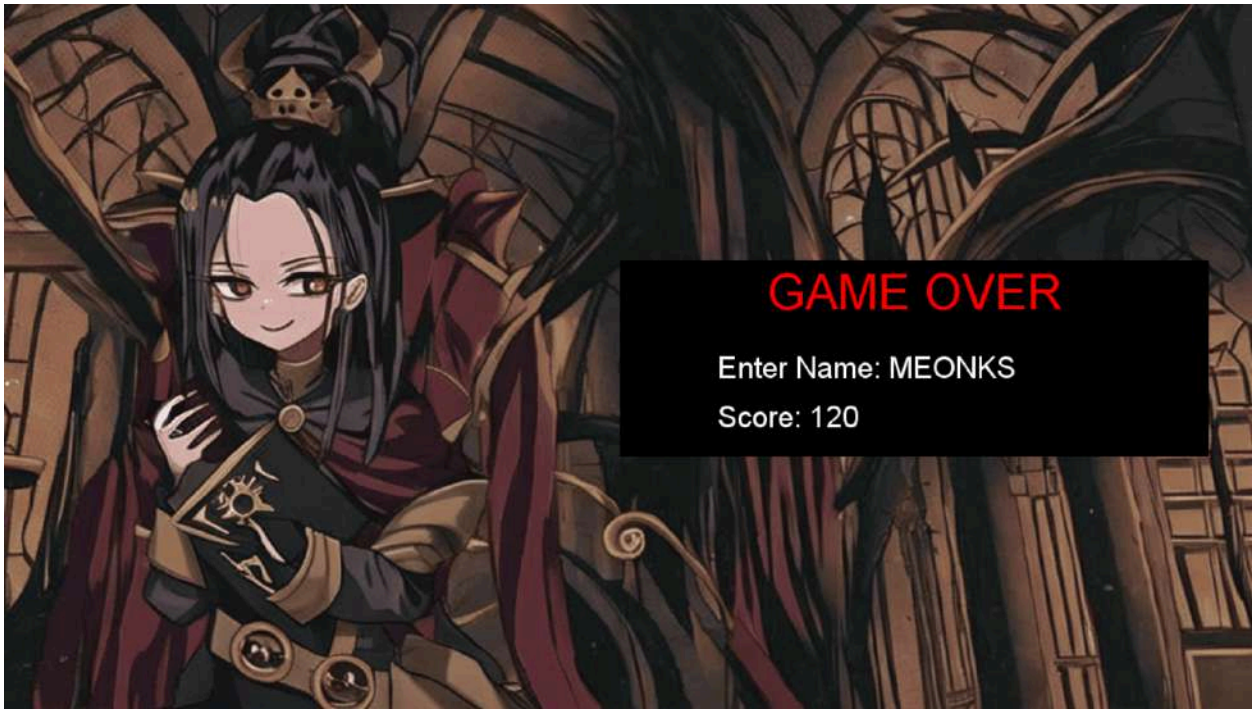

Ketika pemain berhasil menyelesaikan tantangan *minigame* dari wayang hitam, pemain akan langsung dihadapi dengan *BossFight.* Di sini pemain diharuskan untuk mengalahkan boss yang lebih sulit daripada menghadapi musuh biasa. Boss memiliki *HealthPoint* yang lebih tinggi dan dengan ukuran yang lebih besar, memudahkan boss dalam menyerang pemain.

***Victory***



Setelah pemain berhasil mengalahkan boss terakhir, pemain akan dibawa ke menu *Victory* di mana ini menandakan pemain berhasil menamatkan *game GatotRider*. Pemain juga diminta untuk memasukkan *username* dan ditampilkan jumlah *score* yang berhasil diraih.

***Game Over***




Apabila pemain kehabisan *HealthPoint*, pemain akan dibawa ke menu *GameOver* yang menandakan *game* selesai dan pemain gagal menyelesaikan permainan. Pemain akan diminta untuk memasukkan *username* dan akan ditampilkan jumlah *score* yang berhasil didapatkan.

# PENUTUP

Berbagai elemen pemrograman Java berbasis object-oriented programming (OOP), seperti penggunaan kelas, objek, dan mekanisme animasi berhasil diimplementasikan terhadap salah satu platform pengembangan game berbasis Java, yaitu Greenfoot dengan pembuatan game Gatot Rider. Penjelasan kode Java dalam laporan ini bertujuan untuk memberikan gambaran mengenai logika permainan, mulai dari kontrol pemain, interaksi dengan musuh, hingga sistem skor dan nyawa.

Melalui pengerjaan proyek ini, diperoleh pemahaman mendalam mengenai proses pengembangan game, termasuk bagaimana merancang gameplay yang menarik, mempelajari konsep modifier yang digunakan untuk mengatur aksesibilitas dan perilaku kelas, atribut, maupun metode, sehingga mempermudah pengorganisasian kode yang rapi dan terstruktur, konsep parameter yang memungkinkan kami untuk membuat fungsi atau metode yang lebih dinamis dengan kemampuan menerima input sesuai kebutuhan, serta konsep array yang sangat membantu dalam pengelolaan data yang berulang, seperti animasi pada karakter, di mana setiap gambar disimpan dalam sebuah array untuk diakses secara efisien. Dengan mekanisme sederhana namun mendidik, game ini diharapkan dapat memperkenalkan budaya Indonesia kepada pemain, khususnya anak-anak.

Diharapkan agar laporan ini dapat menjadi referensi yang bermanfaat, baik bagi pengembangan game serupa maupun sebagai inspirasi untuk meningkatkan kreativitas di masa mendatang. Kritik dan saran sangat diharapkan untuk memperbaiki kualitas game dan laporan di proyek berikutnya.