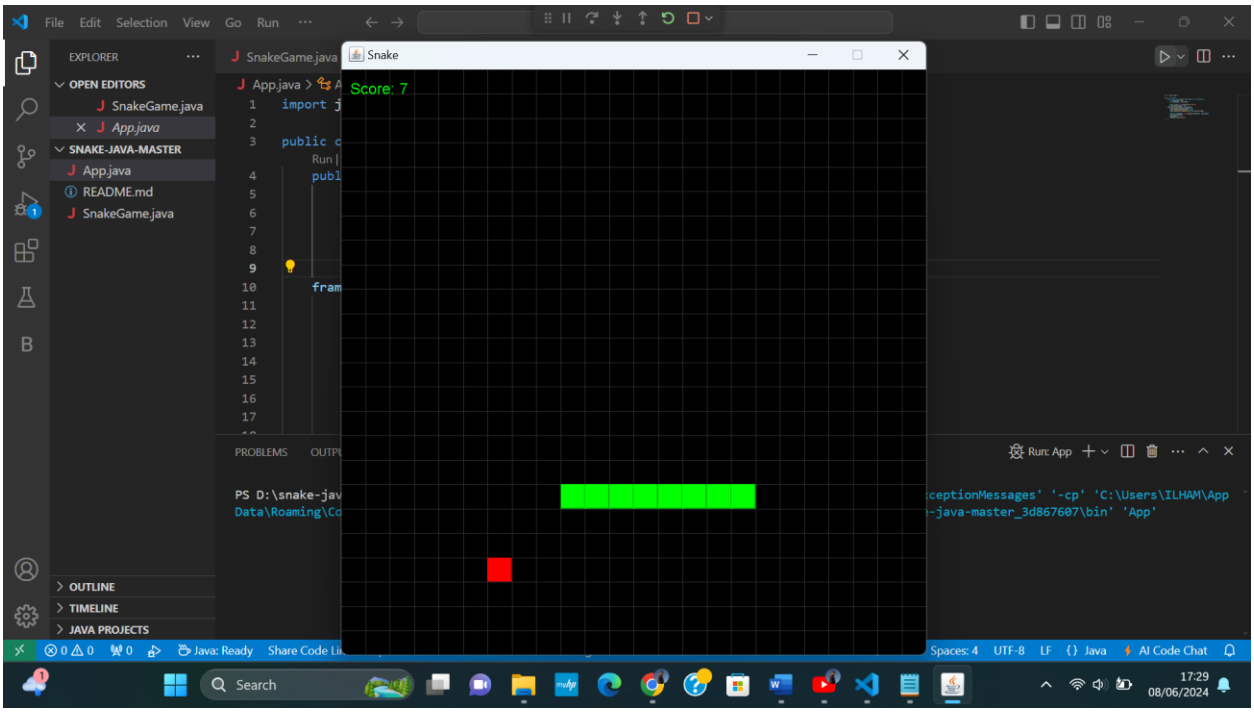
Nama : Ilham Hakiki

NIM : 2222105090

Kelas : 2TI03

Pembuatan Game Snake di Java

Game Snake adalah permainan klasik di mana pemain mengendalikan seekor ular yang bergerak di sekitar papan permainan, mengumpulkan makanan, dan tumbuh semakin panjang. Tujuan utama permainan adalah untuk mengumpulkan sebanyak mungkin makanan tanpa menabrak dinding atau tubuh ular itu sendiri, yang akan menyebabkan permainan berakhir.



Fitur Utama:

1. Papan Permainan:

- Papan permainan adalah area persegi tempat ular bergerak.

- Papan dibagi menjadi grid berukuran 25x25 piksel untuk membantu mengatur posisi ular dan makanan.

2. Ular:

- Ular dimulai dengan satu bagian (kepala) dan tumbuh lebih panjang setiap kali makan makanan.

- Ular digerakkan menggunakan tombol panah pada keyboard (atas, bawah, kiri, kanan).

- Jika ular menabrak dirinya sendiri atau dinding, permainan berakhir.

3. Makanan:

- Makanan muncul di lokasi acak di papan permainan.

- Ketika ular memakan makanan, tubuh ular bertambah panjang dan makanan baru muncul di lokasi acak lainnya.

4. Pengaturan Kecepatan:

- Kecepatan permainan diatur menggunakan `Timer` yang mengatur interval pembaruan permainan setiap 100 milidetik.

5. Skor:

- Skor permainan dihitung berdasarkan panjang tubuh ular, yaitu jumlah makanan yang telah dimakan.

- Skor ditampilkan di layar selama permainan berlangsung.

Komponen Kode Utama:

`App.java`:

- Mengatur jendela aplikasi menggunakan `JFrame`.

- Menginisialisasi objek `SnakeGame` dan menambahkannya ke jendela.

`SnakeGame.java

- Mengatur logika permainan dan tampilan grafis menggunakan `JPanel`.

- Mengelola pergerakan ular, deteksi tabrakan, dan penempatan makanan.

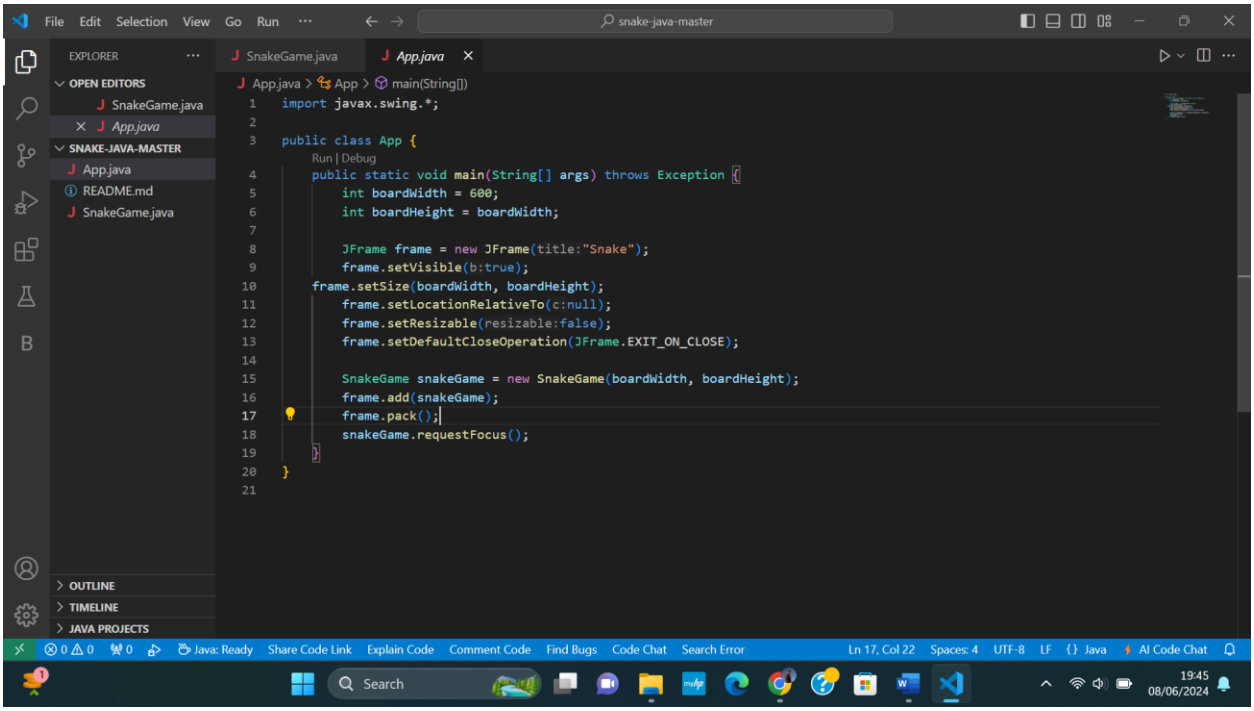
- Mendefinisikan kontrol permainan dengan mendeteksi input keyboard.

- Memperbarui dan menggambar ulang papan permainan dalam interval waktu tertentu.

Dengan permainan ini, pemain dapat menikmati tantangan mengendalikan ular yang semakin panjang sambil menghindari tabrakan untuk mencapai skor setinggi mungkin.

Projek ini adalah implementasi game Snake sederhana menggunakan Java dengan Swing untuk GUI. Projek ini terdiri dari dua file utama: `App.java` dan `SnakeGame.java`. Pejelasan kedua filenya :

App.java



```
import javax.swing.*;
```

- Mengimpor pustaka `javax.swing` untuk menggunakan komponen GUI seperti `JFrame`.

```
public class App {
    public static void main(String[] args) throws Exception {
        int boardWidth = 600;
        int boardHeight = boardWidth;
```

- Mendeklarasikan kelas `App` dan metode `main`.
- Mengatur lebar dan tinggi papan permainan menjadi 600 piksel.

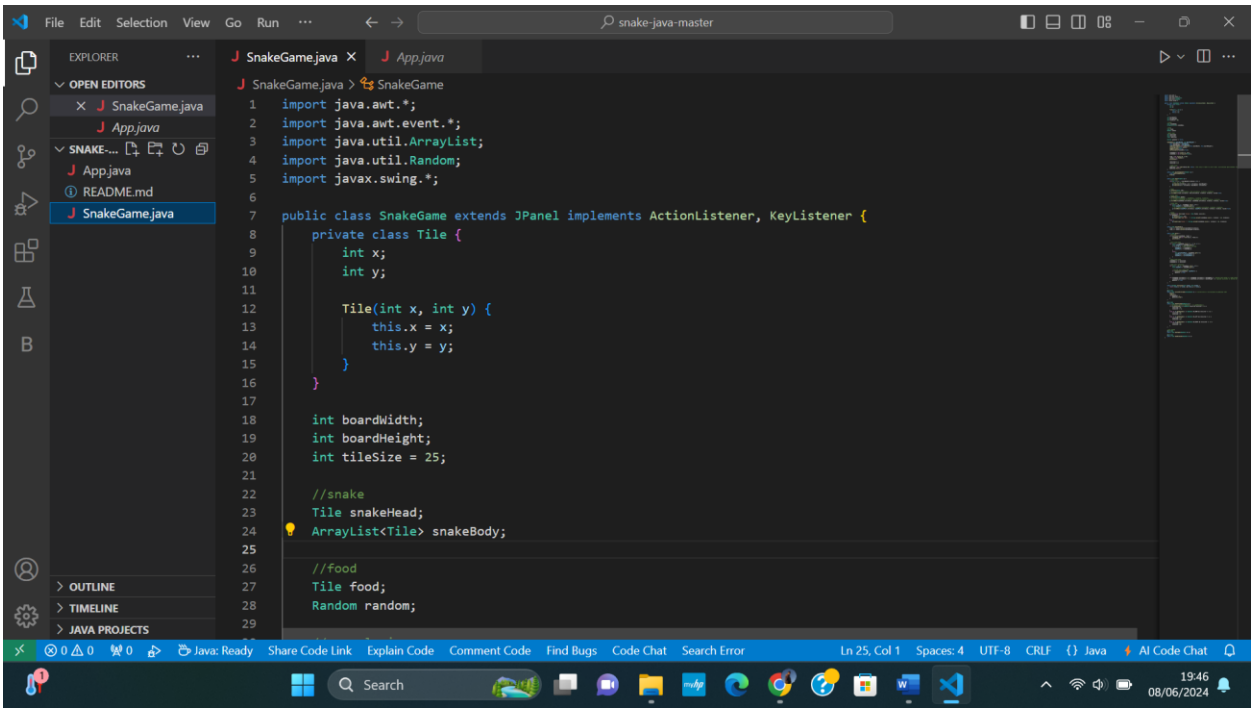
```
        JFrame frame = new JFrame("Snake");
        frame.setVisible(true);
        frame.setSize(boardWidth, boardHeight);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- Membuat jendela aplikasi dengan judul "Snake".
- Mengatur jendela agar terlihat, ukurannya sesuai dengan lebar dan tinggi papan, dan posisinya di tengah layar.
- Membuat jendela tidak dapat diubah ukurannya dan menutup aplikasi saat jendela ditutup.

```
        SnakeGame snakeGame = new SnakeGame(boardWidth, boardHeight);
        frame.add(snakeGame);
        frame.pack();
        snakeGame.requestFocus();
    }
}
```

- Membuat instance `SnakeGame` dengan lebar dan tinggi papan permainan.
- Menambahkan `SnakeGame` ke dalam jendela `JFrame`.
- Mengatur ukuran jendela sesuai dengan konten dan mengatur fokus pada `SnakeGame`.

SnakeGame.java




```
import java.awt.*;
```

```
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.*;
```

- Mengimpor pustaka untuk grafis, event handling, array list, random number generation, dan komponen GUI.

```
public class SnakeGame extends JPanel implements ActionListener, KeyListener {
    private class Tile {
        int x;
        int y;

        Tile(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
```

- Mendeklarasikan kelas `SnakeGame` yang memperluas `JPanel` dan mengimplementasikan `ActionListener` dan `KeyListener`.
- Mendeklarasikan kelas dalam `Tile` untuk mewakili bagian dari ular atau makanan dengan koordinat x dan y.

```
    int boardWidth;
```

int boardHeight;

int tileSize = 25;

- Mendeklarasikan variabel untuk lebar papan, tinggi papan, dan ukuran tile.

Tile snakeHead;

ArrayList<Tile> snakeBody;

- Mendeklarasikan variabel untuk kepala ular dan tubuh ular sebagai daftar tile.

Tile food;

Random random;

- Mendeklarasikan variabel untuk makanan dan objek random untuk penempatan makanan.

int velocityX;

int velocityY;

Timer gameLoop;

boolean gameOver = false;

- Mendeklarasikan variabel untuk kecepatan ular dalam arah x dan y, timer untuk loop permainan, dan status game over.

SnakeGame(int boardWidth, int boardHeight) {

this.boardWidth = boardWidth;

this.boardHeight = boardHeight;

setPreferredSize(new Dimension(this.boardWidth, this.boardHeight));

setBackground(Color.black);

addKeyListener(this);

setFocusable(true);

- Konstruktor untuk `SnakeGame` menginisialisasi lebar dan tinggi papan, mengatur ukuran preferensi, warna latar belakang, menambahkan key listener, dan membuat panel fokusable.

snakeHead = new Tile(5, 5);

snakeBody = new ArrayList<Tile>();

food = new Tile(10, 10);

random = new Random();

placeFood();

velocityX = 1;

velocityY = 0;

- Menginisialisasi posisi awal kepala ular dan tubuh ular sebagai array list kosong.
- Menginisialisasi posisi awal makanan dan objek random untuk penempatan makanan.
- Mengatur kecepatan awal ular bergerak ke kanan.

gameLoop = new Timer(100, this);

gameLoop.start();

}

- Menginisialisasi timer untuk loop permainan dengan interval 100 milidetik dan memulai timer.

public void paintComponent(Graphics g) {

super.paintComponent(g);

draw(g);

}

- Metode untuk mengecat ulang komponen. Memanggil `super.paintComponent(g)` untuk membersihkan layar dan kemudian memanggil metode `draw`.

public void draw(Graphics g) {

for (int i = 0; i < boardWidth / tileSize; i++) {

*g.drawLine(i * tileSize, 0, i * tileSize, boardHeight);*

*g.drawLine(0, i * tileSize, boardWidth, i * tileSize);*

}

- Menggambar garis-garis grid di papan permainan.

g.setColor(Color.red);

*g.fill3DRect(food.x * tileSize, food.y * tileSize, tileSize, tileSize, true);*

- Menggambar makanan dengan warna merah.

g.setColor(Color.green);

```
    g.fill3DRect(snakeHead.x * tileSize, snakeHead.y * tileSize, tileSize, tileSize, true);
```

- Menggambar kepala ular dengan warna hijau.

```
    for (int i = 0; i < snakeBody.size(); i++) {
        Tile snakePart = snakeBody.get(i);
        g.fill3DRect(snakePart.x * tileSize, snakePart.y * tileSize, tileSize, tileSize, true);
    }
```

- Menggambar tubuh ular dengan warna hijau.

```
    g.setFont(new Font("Arial", Font.PLAIN, 16));
    if (gameOver) {
        g.setColor(Color.red);
        g.drawString("Game Over: " + String.valueOf(snakeBody.size()), tileSize - 16, tileSize);
    } else {
        g.drawString("Score: " + String.valueOf(snakeBody.size()), tileSize - 16, tileSize);
    }
}
```

- Mengatur font dan menggambar skor atau pesan game over di layar.

```
public void placeFood() {
    food.x = random.nextInt(boardWidth / tileSize);
    food.y = random.nextInt(boardHeight / tileSize);
}
```

- Menempatkan makanan di lokasi acak pada papan permainan.

```
public void move() {
    if (collision(snakeHead, food)) {
        snakeBody.add(new Tile(food.x, food.y));
        placeFood();
    }
```

- Menggerakkan ular dan menambahkan bagian baru ke tubuh jika makanan dimakan. Menempatkan makanan baru di lokasi acak.

```
    for (int i = snakeBody.size() - 1; i >= 0; i--) {
        Tile snakePart = snakeBody.get(i);
        if (i == 0) {
            snakePart.x = snakeHead.x;
            snakePart.y = snakeHead.y;
        } else {
            Tile prevSnakePart = snakeBody.get(i - 1);
            snakePart.x = prevSnakePart.x;
            snakePart.y = prevSnakePart.y;
        }
    }
```

- Menggerakkan tubuh ular mengikuti kepala ular.

```
    snakeHead.x += velocityX;
    snakeHead.y += velocityY;
```

- Menggerakkan kepala ular berdasarkan kecepatan saat ini.

```
    for (int i = 0; i < snakeBody.size(); i++) {
        Tile snakePart = snakeBody.get(i);
        if (collision(snakeHead, snakePart)) {
            gameOver = true;
        }
    }
```

- Mengecek apakah kepala ular menabrak tubuhnya sendiri. Jika ya, permainan berakhir.

```
    if (snakeHead.x * tileSize < 0 || snakeHead.x * tileSize > boardWidth ||
        snakeHead.y * tileSize < 0 || snakeHead.y * tileSize > boardHeight) {
        gameOver = true;
    }
}
```

- Mengecek apakah kepala ular menabrak dinding papan permainan. Jika ya, permainan berakhir.

```
public boolean collision(Tile tile1, Tile tile2) {
    return tile1.x == tile2.x && tile1.y == tile2.y;
}
```

- Metode untuk mengecek apakah dua tile bertabrakan.

```
@Override
public void actionPerformed(ActionEvent e) {
    move();
    repaint();
    if (gameOver) {
        gameLoop.stop();
    }
}
```

- Dipanggil oleh timer setiap 100 milidetik. Menggerakkan ular, mengecat ulang komponen, dan menghentikan permainan jika game over.

```
@Override
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_UP && velocityY != 1) {
        velocityX = 0;
        velocityY = -1;
    } else if (e.getKeyCode() == KeyEvent.VK_DOWN && velocityY != -1) {
        velocityX = 0;
        velocityY = 1;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT && velocityX != 1) {
        velocityX = -1;
```

```
        velocityY = 0;
    } else if (e.getKeyCode() == KeyEvent.VK_RIGHT && velocityX != -1) {
        velocityX = 1;
        velocityY


= 0;
    }
  }
```

- Menangani input dari keyboard untuk mengatur arah pergerakan ular.

```
  @Override
  public void keyTyped(KeyEvent e) {}
  @Override
  public void keyReleased(KeyEvent e) {}
}
```

- Metode yang diperlukan untuk `KeyListener` tetapi tidak digunakan.

Kesimpulan

Kode di atas membuat permainan Snake sederhana menggunakan Java dan Swing. Permainan ini memiliki komponen utama yaitu ular yang dapat bergerak di dalam papan permainan, makanan yang muncul secara acak, dan skor yang meningkat setiap kali ular memakan makanan. Permainan berakhir ketika ular menabrak dinding atau dirinya sendiri. Aplikasi ini menggunakan konsep dasar seperti pengaturan GUI, event handling, dan timer untuk mengelola loop permainan.