LAPORAN PROYEK PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA 2024

IMPLEMENTASI STRUKTUR DATA PADA GAME CRIME SOLVER: PETUALANGAN DETEKTIF MISTERI DI BALIK LOKASI

Oleh Kelompok 26

Anggota:

Pradiva Wiyudha Maheswara	**F1D022151**
Julia Ariyani	**F1D02310062**
Kania Putri Rohimatuz Azzahra H	**F1D02310065**
Lalu Ahmad Purwadi	**F1D02310115**

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MATARAM

2024

LEMBAR PENGESAHAN LAPORAN PROYEK
PRAKTIKUM ALGORITMA DAN STRUKTUR DATA 2024

1.	Kelompok	:	26
2.	Judul Proyek	:	Implementasi Struktur Data pada Game Crime Solver: Petualangan Detektif Misteri Di Balik Lokasi
3.	Anggota Kelompok	:	Pradiva Wiyudha Maheswara (F1D022151)
			Julia Ariyani (F1D02310062)
			Kania Putri Rohimatuz A. H. (F1D02310065)
			Lalu Ahmad Purwadi (F1D02310115)

Laporan proyek ini disusun sesuai dengan kaidah penyusunan yang telah ditentukan dan dibuat sebagai syarat mata kuliah Algoritma dan Struktur Data 2024.

Mataram, 22 Desember 2024

Telah diperiksa dan disahkan oleh:

Koordinator Asisten	**Asisten Pembimbing**
Aditya Rahmatdiyansyah	**Teguh.....**
F1D022031	**F1D022...**

1. 1 Judul Proyek

Judul proyek kami yaitu "*Crime Solver* :Petualangan Detektif Misteri di Balik Lokasi"

1. 2 Latar Belakang

Pemahaman terhadap struktur data dan algoritma adalah fondasi penting dalam dunia pemrograman dan pengembangan perangkat lunak. Struktur data seperti *linked list*, *stack*, *queue*, *sorting*, *searching*, dan *graph* merupakan elemen kunci yang mendukung pengelolaan data secara efisien dan efektif. *Linked list*, misalnya, memungkinkan pengelolaan data secara dinamis melalui elemen yang saling terhubung, sehingga cocok digunakan dalam skenario yang membutuhkan fleksibilitas seperti sistem navigasi atau pengelolaan daftar tugas. *Stack* dan *queue* memiliki peran penting dalam mengatur data berdasarkan prinsip tertentu, di mana *stack* menerapkan metode *Last In First Out* (LIFO) yang sering digunakan untuk operasi *undo-redo* atau manajemen fungsi rekursif, sedangkan *queue* menggunakan metode *First In First Out* (FIFO) yang umum diterapkan pada antrian tugas atau sistem penjadwalan.

Selain itu, algoritma *sorting* dan *searching* sangat krusial dalam pengolahan data untuk mendapatkan hasil yang cepat dan akurat. *Sorting* membantu menyusun data dalam urutan tertentu sehingga mempermudah proses pencarian atau analisis, sedangkan *searching* digunakan untuk menemukan data tertentu dalam sekumpulan informasi dengan berbagai metode seperti linear *search* atau *binary search*. Tidak kalah penting, *graph* sebagai representasi dari hubungan antar elemen, sering digunakan dalam berbagai aplikasi, seperti pencarian jalur terpendek pada peta atau pengelolaan jaringan sosial. Dengan memahami dan menerapkan konsep-konsep ini, pengembang dapat menciptakan solusi yang lebih optimal untuk menyelesaikan masalah-masalah kompleks dalam berbagai bidang. Pengembang perangkat lunak dapat menciptakan sistem yang tidak hanya berfungsi dengan baik tetapi juga mampu menangani data dalam jumlah besar dengan kecepatan yang memadai. Hal ini menjadi dasar bagi pengembangan berbagai aplikasi modern, mulai dari sistem navigasi, kecerdasan buatan, hingga game interaktif yang membutuhkan pengelolaan data yang kompleks.

Game edukasi berbasis logika dan pemrograman semakin diminati karena dapat memberikan pengalaman bermain yang interaktif sekaligus mengasah kemampuan berpikir analitis pemain. "*Crime Solver*: Petualangan Detektif Misteri di Balik Lokasi" dikembangkan

untuk memenuhi kebutuhan akan game yang tidak hanya menghibur, tetapi juga mendidik, khususnya dalam konteks penerapan struktur data dan algoritma dalam pemrograman.

1.3 Deskripsi Program

Game "*Crime Solver*: Petualangan Detektif Misteri di Balik Lokasi" adalah sebuah permainan petualangan interaktif yang mengajak pemain untuk memecahkan misteri kriminal di berbagai lokasi dengan menerapkan struktur data seperti *linked list*, *stack*, *queue*, *sorting*, *searching*, dan *graph*. Pemain berperan sebagai seorang detektif yang harus mengumpulkan memecahkan kasus criminal dengan cara mengunjungi berbagai Lokasi, mengumpulkan petunjuk, dan mengidentifikasi tersangka yang tepat.

Dalam permainan ini, beberapa sturktur data sudah diimplementasikan untuk mendukung berjalannya *game* "*Crime Solver*: Petualangan Detektif Misteri di Balik Lokasi". *Linkedlist* digunakan untuk menyimpan data akun dan *queue* untuk melacak akun pengguna yang sedang *log-in*. Untuk sistem dari lokasi pada *game* menggunakan *graph* yang digunakan untuk menyambungkan lokasi, sehingga pemain dapat berpindah antar lokasi yang saling terhubung dan mengumpulkan petunjuk di setiap lokasi. Kemudian *stack* digunakan untuk menyimpan jejeak perjalanan dari pemain, sehingga pemain dapat melihat lokasi mana saja yang sudah di kunjungin selam bermain.

Selain itu, dalam *game* "*Crime Solver*: Petualangan Detektif Misteri di Balik Lokasi" terdapat teka-teki yang diimplementasikan dengan menggunakan *binary tree*. Pemain dapat memilih diantara dua pilihan dari teka-teki dalam permainan yang dapat mempengaruhi alur dari "*Crime Solver*: Petualangan Detektif Misteri di Balik Lokasi". Sistem skor juga diterapkan dalam permainan yang dimulai dari 100 poin. Setiap kesalahan pada permainan akan dikenai penalty 30 poin dan permainan akan berakhir jika skor mencapai 0 poin. Namun, pemain akan mendapatkan tambahan 20 poin jika berhasil menebak tersangka dengan benar.

Alur *game* ini akan dimulai dengan pemain yang diminta untuk membuat akun terlebih dahulu dan *log-in* dengan akun tersebut. Setelah berhasil *log-in*, pemain dapat mulai bermain dengan mengunjungi berbagai lokasi yang tersedia seperti TKP, Kamar, Dapur, Taman, Garasi, dan Gudang. Pada setiap lokasi, terdapat beberapa petunjuk yang dapat membantu pemain dalam mengungkap kasus. Pemain juga dapat memecahkan teka-teki pada permainan dan memilih tersangka yang dicurigai sebagai pelaku kejahatan.

1.4 Algoritma

Berikut ini merupakan algoritma yang digunakan untuk menyelesaikan permasalahan pada program:

1. Program dimulai dengan meminta pemain untuk membuat akun terlebih dahulu.
2. Setelah pemain membuat akun, maka selanjutnya pemain bisa masuk dengan menggunakan akun yang telah dibuat tadi.
3. Program menampilkan skor dan menu dari *game Crime Solver* dengan pilihan kunjungi lokasi, lihat jejak perjalanan, lihat semua lokasi, selesaikan teka-teki, selesaikan kasus, kemudian yang terakhir yaitu keluar.
4. Jika pemain memilih untuk mengunjungi lokasi
    a. Menampilkan daftar lokasi yang sudah tersedia.
    b. Pemain memasukkan lokasi yang ingin dikunjungi, jika lokasi ditemukan maka pemain akan diarahkan untuk lanjut ke penyeledikan kasus dari game *Crime Solver.*
    c. Jika tidak valid maka akan ditampilkan "Lokasi tidak ditemukan".
5. Jika pemain memilih untuk melihat jejak perjalanan maka lokasi yang sudah di kunjungi oleh pemain akan ditampilkan.
6. Jika pemain memilih untuk melihat daftar dari lokasi, maka daftar lengkap lokasi akan ditampilkan.
7. Jika pemain memilih untuk menyelesaikan teka-teki
    a. Teka-teki dimulai dari akar yang setiap langkahnya menampilkan petunjuk dan dua opsi jawaban.
    b. Pemain memasukkan pilihan berdasarkan dua opsi jawaban yang tersedia.
    c. Jika valid maka akan lanjut ke langkah berikutnya, namun jika tida valid maka program akan meminta agar pemain memasukkan ulang opsi jawabannya.
    d. Kemudian jika teka-teki sudah selesai dipecahkan oleh pemain maka akan menampilkan "Teka-teki selesai".
8. Jika pemain memilih untuk mencari tersangka
    a. Pemain akan diminta untuk memilih tersangka mana yang pemain curigai sebagai pelaku.
    b. Kemudian pemain akan diarahkan kembali ke menu untuk menyelesaikan kasus utnuk melihat apakah benar tebakan tersangka dari pemain.

c. Jika benar maka pemain akan mendapat skor tambahan 20 poin
d. Jika tidak maka pemain akan mendapat pengurangan skor sebesar 30 poin

9. Jika pemain memilih untuk menyelsaikan kasus tanpa memilih tersangka terlebih dahulu, maka pemain akan dikenai pengurangan poin dan pemain diarahkan ke menu utama untuk bermain.
10. Jika pemain memilih untuk keluar maka akan menampilkan "Terima kasih telah bermain".
11. Program selesai.

1. 5 Penjelasan Kode

1.5.1 *Crime Solver Class*

```
public class CrimeSolver {
public static void main(String[] args) {
System.out.println("=== Selamat datang di permainan Crime Solver
    ===");
System.out.println("Silakan login untuk memulai permainan.");
CrimeSolverLogin loginSystem = new CrimeSolverLogin();
loginSystem.mainMenu();
AyoMain gameSystem = new AyoMain();
gameSystem.startGame();
}
}
```

Script di atas merupakan bagian dari program untuk permainan. Program ini diawali dengan menampilkan pesan sambutan kepada pemain melalui fungsi "`System.out.println`", lalu mengarahkan pemain untuk login sebelum memulai permainan. Program ini memanfaatkan dua kelas, yaitu "`CrimeSolverLogin`" dan "`AyoMain`". Kelas `CrimeSolverLogin` dipanggil melalui metode "`mainMenu()`" untuk menangani proses *login* pemain. Setelah *login* selesai, kelas "`AyoMain`" dipanggil melalui metode "`startGame()`" untuk memulai permainan utama.

1.5.2 *Crime Solver Login Class*

```
import java.util.Scanner;

class CrimeSolverLogin {
private QueueLogin queueLogin;
private Account head;

public CrimeSolverLogin() {
queueLogin = new QueueLogin();
}
static class Account {
String username;
```

```
String password;
Account next;

public Account(String username, String password) {
this.username = username;
this.password = password;
this.next = null;
}
}
```

Scirpt di atas merupakan bagian dari implementasi kelas "`CrimeSolverLogin`" dalam Java, yang bertanggung jawab menangani proses *login* pemain untuk permainan *Crime Solver*. Kelas ini memiliki dua atribut utama: "`queueLogin`", yang merupakan objek dari kelas "`QueueLogin`" dan "`head`", yang berfungsi sebagai pointer ke awal daftar akun pengguna yang tersimpan dalam struktur data *linked list*. Di dalam kelas ini, terdapat juga kelas statis "`Account`" yang merepresentasikan akun pengguna, dengan atribut "`username`" dan "`password`" untuk menyimpan kredensial pengguna, serta atribut "`next`" yang menunjuk ke akun berikutnya dalam *linked list*. Konstruktor "`Account`" digunakan untuk menginisialisasi data akun, sedangkan konstruktor utama "`CrimeSolverLogin`" menginisialisasi objek "`queueLogin`".

```
public void signUp(String username, String password) {
if (findAccount(username) != null) {
System.out.println("Username sudah digunakan. Silakan gunakan username
       lain.");
return;
}
Account newAccount = new Account(username, password);
if (head == null) {
head = newAccount;
} else {
Account current = head;
while (current.next != null) {
current = current.next;
}
current.next = newAccount;
}
System.out.println("Akun berhasil dibuat! Silakan login.");
}
```

Script "`signUp`" dalam kelas "`CrimeSolverLogin`" berfungsi untuk mendaftarkan akun baru ke dalam sistem. Metode ini menerima dua parameter, yaitu "`username`" dan "`password`", untuk membuat akun baru. Pertama, metode ini memeriksa apakah "`username`" yang dimasukkan sudah digunakan dengan memanggil metode "`findAccount`". Jika "`findAccount`" menemukan akun dengan "`username`" yang

sama, pengguna diberi notifikasi bahwa "`username sudah digunakan`", dan proses pendaftaran dihentikan. Jika username belum digunakan, objek "`Account`" baru dibuat dengan kredensial yang diberikan. Jika daftar akun masih kosong (ditunjukkan dengan "`head`" bernilai "`null`"), akun baru menjadi "`head`".

```
public boolean login(String username, String password) {
Account account = findAccount(username);
if (account != null && account.password.equals(password)) {
System.out.println("Login berhasil! Selamat datang, " + username +
      "!");
queueLogin.enqueue(username);
return true;
}
System.out.println("Login gagal. Username atau password salah.");
return false;
}
private Account findAccount(String username) {
Account current = head;
while (current != null) {
if (current.username.equals(username)) {
return current;
}
current = current.next;
}
return null;
}
```

Script "`login`" dalam kelas "`CrimeSolverLogin`" memverifikasi autentikasi pengguna berdasarkan "`username`" dan "`password`" dengan menggunakan metode "`findAccount`", yang menelusuri *linked list* dari "`head`" untuk mencari akun yang cocok. Jika akun ditemukan dan *password* sesuai, proses *login* berhasil, pesan sukses ditampilkan, dan "`username`" pengguna ditambahkan ke antrian *login* menggunakan metode "`enqueue`" dari objek "`queueLogin`". Sebaliknya, jika akun tidak ditemukan atau *password* salah, proses *login* gagal dan pesan kesalahan ditampilkan. Dengan pendekatan ini, metode login memastikan hanya pengguna dengan kredensial valid yang dapat mengakses sistem.

```
public void mainMenu() {
Scanner scanner = new Scanner(System.in);
while (true) {
System.out.println("\n=== Crime Solver Login ===");
System.out.println("1. Sign-Up");
System.out.println("2. Login");
System.out.println("3. Lihat Akun yang Login");
```

```
System.out.println("4. Keluar");
System.out.print("Pilihan: ");
int choice = -1;
try {
choice = Integer.parseInt(scanner.nextLine());
} catch (NumberFormatException e) {
System.out.println("Input tidak valid. Silakan pilih angka antara 1
       dan 4.");
continue;
}
switch (choice) {
case 1:
System.out.print("Username: ");
String username = scanner.nextLine();
System.out.print("Password: ");
String password = scanner.nextLine();
signUp(username, password);
break;
case 2:
System.out.print("Username: ");
username = scanner.nextLine();
System.out.print("Password: ");
password = scanner.nextLine();
if (login(username, password)) {
AyoMain gameSystem = new AyoMain();
gameSystem.startGame();
}
break;
case 3:
queueLogin.displayLoginAccounts();
break;
case 4:
System.out.println("Terima kasih!");
scanner.close();
return;
default:
System.out.println("Pilihan tidak valid.");
}
```

```
}
}
}
```

Script "`mainMenu`" dalam kelas "`CrimeSolverLogin`" adalah antarmuka utama yang memungkinkan pengguna berinteraksi dengan sistem melalui menu berbasis teks. Metode ini menggunakan perulangan tak terbatas "`while (true)`" untuk terus menampilkan opsi menu hingga pengguna memilih untuk keluar. Menu menyediakan empat pilihan utama: 1."`Sign-Up`", untuk mendaftarkan akun baru menggunakan metode "`signUp`;" 2. "`Login`", untuk masuk ke sistem dengan memverifikasi *username* dan *password* menggunakan metode *login*, dan jika berhasil, permainan dimulai melalui kelas "`AyoMain`;" 3. Lihat Akun yang "`Login`", untuk menampilkan daftar akun yang telah *login* dengan memanggil metode "`displayLoginAccounts`" dari objek "`queueLogin`;" serta 4."`Keluar`," untuk keluar dari program dengan menutup *scanner*

1.5.3 *Queue Login Class*

```
class Nodequeue {
String username;
Nodequeue next;

public Nodequeue(String username) {
this.username = username;
this.next = null;
}
}
```

Kelas "`Nodequeue`" merepresentasikan simpul dalam antrean dinamis dengan dua atribut: "`username`" untuk menyimpan data pengguna dan "`next`" sebagai referensi ke simpul berikutnya. Konstruktor menerima parameter "`username`" untuk inisialisasi data, sementara "`next`" di *set* ke "`null`", menandakan simpul belum terhubung.

```
class QueueLogin {
private Nodequeue front, rear;
private int size;

public QueueLogin() {
front = rear = null;
size = 0;
}
public void enqueue(String username) {
Nodequeue newNode = new Nodequeue(username);
if (rear == null) {
front = rear = newNode;
} else {
rear.next = newNode;
```

```
rear = newNode;
}
size++;
System.out.println("Akun " + username + " berhasil ditambahkan.");
}
```

Kelas "`QueueLogin`" adalah implementasi antrean dinamis dengan atribut "`front`" (simpul depan), "`rear`" (simpul belakang), dan "`size`" (jumlah elemen). Konstruktor menginisialisasi "`front`" dan "`rear`" ke "`null`" serta "`size`" ke "`0`". Metode "`enqueue`" menambahkan elemen ke belakang antrean dengan membuat simpul baru menggunakan data *username*. Jika antrean kosong, "`front`" dan "`rear`" menunjuk ke simpul baru; jika tidak, simpul baru dihubungkan ke "`rear`", lalu "`rear`" diperbarui. Ukuran antrean bertambah, dan pesan keberhasilan ditampilkan.

```
public String dequeue() {
if (isEmpty()) {
System.out.println("Queue kosong, tidak ada akun yang dapat
       dikeluarkan!");
return null;
}
String username = front.username;
front = front.next;
if (front == null) {
rear = null;
}
size--;
System.out.println("Akun " + username + " telah dikeluarkan dari
       queue.");
return username;
}
```

Metode "`dequeue()`" pada kelas "`QueueLogin`" digunakan untuk mengeluarkan elemen dari depan antrean. Jika antrean kosong (ditentukan dengan metode "`isEmpty()`"), maka pesan peringatan akan ditampilkan dan metode mengembalikan "`null`". Jika antrean tidak kosong, elemen pertama (data *username*) akan disalin ke dalam variabel, kemudian "`front`" diperbarui untuk menunjuk ke simpul berikutnya. Jika setelah pengeluaran, "`front`" menjadi "`null`" (antrean kosong), maka "`rear`" juga di *set* "`null`".

```
public void displayLoginAccounts() {
if (isEmpty()) {
System.out.println("Tidak ada akun yang login.");
return;
}
System.out.println("Akun yang sudah login:");
Nodequeue current = front;
while (current != null) {
```

```
System.out.println("- " + current.username);
current = current.next;
}
}
public boolean isEmpty() {
return size == 0;
}
public int getSize() {
return size;
}
}
```

Metode "displayLoginAccounts()" menampilkan semua akun yang *login* dalam antrean. Jika antrean kosong (diperiksa oleh "isEmpty()"), pesan "Tidak ada akun yang login" ditampilkan. Jika antrean tidak kosong, daftar akun yang *login* akan ditampilkan dengan mengiterasi melalui setiap simpul, mulai dari "front" hingga "rear". Metode "isEmpty()" memeriksa apakah antrean kosong dengan mengembalikan *true* jika ukuran antrean "size" adalah "0". Sedangkan "getSize()" mengembalikan nilai "size" untuk mengetahui jumlah elemen dalam antrean.

1.5.4 Teka Teki *Class*

```
import java.util.Scanner;
class Teka {
String clue;
Teka opsi1;
Teka opsi2;
String opsi1Text;
String opsi2Text;

public Teka(String clue, String opsi1Text, String opsi2Text) {
this.clue = clue;
this.opsi1Text = opsi1Text;
this.opsi2Text = opsi2Text;
this.opsi1 = null;
this.opsi2 = null;
}
}
```

Kelas "Teka" digunakan untuk merepresentasikan sebuah teka-teki yang memiliki dua opsi jawaban. Setiap objek dari kelas ini memiliki atribut "clue" yang menyimpan petunjuk atau soal teka-teki, serta dua atribut "opsi1Text" dan "opsi2Text" yang menyimpan teks untuk dua opsi jawaban. Atribut "opsi1" dan "opsi2" adalah referensi ke objek "Teka" lainnya, yang menghubungkan teka-teki ini ke teka-teki selanjutnya jika salah satu opsi dipilih. Konstruktor kelas ini menerima parameter untuk menginisialisasi "clue", "opsi1Text", dan "opsi2Text", sementara "opsi1" dan

"opsi2" di *set* ke "null", menandakan bahwa opsi-opsi tersebut belum dihubungkan ke teka-teki lain.

```
class TekaTeki {
private final Teka root;
public TekaTeki() {
root = new Teka(
"Kamu menemukan sidik jari di tempat kejadian.",
"Periksa sidik jarinya lebih lanjut.",
"Periksa CCTV untuk petunjuk."
);
root.opsi1 = new Teka(
"Sidik jari cocok dengan salah satu tersangka!",
"Tanya tersangka apakah dia ada di TKP.",
"Cari bukti lebih lanjut di lokasi."
);
root.opsi2 = new Teka(
"CCTV menunjukkan seseorang di sekitar lokasi kejadian.",
"Periksa wajah orang tersebut.",
"Cari petunjuk lain di lokasi."
);
root.opsi1.opsi1 = new Teka(
"Wajah orang tersebut cocok dengan deskripsi saksi.",
"Tanya orang tersebut.",
"Periksa alibi orang tersebut."
);
root.opsi1.opsi2 = new Teka(
"Tidak ada petunjuk lebih lanjut di lokasi.",
"Periksa tempat lain.",
"Lanjutkan penyelidikan."
);
root.opsi2.opsi1 = new Teka(
"Wajah orang tersebut cocok dengan deskripsi saksi.",
"Tanya orang tersebut.",
"Periksa alibi orang tersebut."
);
root.opsi2.opsi2 = new Teka(
"Tidak ada petunjuk lebih lanjut di lokasi.",
"Periksa tempat lain.",
"Lanjutkan penyelidikan."
);
}
```

Kelas "TekaTeki" merupakan implementasi dari teka-teki bercabang, di mana setiap objek "Teka" berfungsi sebagai langkah dalam teka-teki tersebut. Kelas ini memiliki atribut "root", yang merupakan objek "Teka" pertama (awal) dari sebuah cerita atau teka-teki. Konstruktor kelas "TekaTeki()" menginisialisasi "root" dengan teka-teki pertama yang berisi petunjuk dan dua opsi jawaban. Setiap opsi pada "root" mengarah ke objek "Teka" lain, yang masing-masing memiliki dua opsi jawaban yang dapat dipilih, membentuk struktur bercabang. Contohnya, pada "opsi1" dari "root",

jika opsi pertama dipilih, maka akan membawa ke teka-teki baru yang memeriksa kesesuaian wajah tersangka dengan deskripsi saksi, dan seterusnya. Struktur ini memungkinkan alur teka-teki berkembang tergantung pada pilihan yang diambil, dengan tiap opsi mengarah ke teka-teki selanjutnya.

```
public void solve() {
Scanner scanner = new Scanner(System.in);
Teka current = root;

while (current.opsi1 != null || current.opsi2 != null) {
System.out.println(current.clue);
System.out.println("1. " + current.opsi1Text);
System.out.println("2. " + current.opsi2Text);
System.out.print("Pilih 1 atau 2: ");
String jawaban = scanner.nextLine();

if (jawaban.equals("1")) {
current = current.opsi1;
} else if (jawaban.equals("2")) {
current = current.opsi2;
} else {
System.out.println("Pilihan tidak valid, pilih 1 atau 2.");
}
}
System.out.println(current.clue);
System.out.println("Penyelidikan selesai!");
}
}
```

Metode "`solve()`" pada kelas "`TekaTeki`" memungkinkan pengguna untuk memainkan teka-teki dengan memilih antara dua opsi di setiap langkah berdasarkan petunjuk yang diberikan. Dimulai dengan "`root`", yang berisi petunjuk pertama, metode ini terus menampilkan petunjuk dan dua opsi kepada pengguna. Pengguna diminta untuk memilih antara opsi "`1`" atau "`2`" dengan memasukkan angka tersebut. Jika pengguna memilih opsi "`1`", maka "`current`" akan diperbarui untuk menunjuk ke "`opsi1`;" jika memilih opsi "`2`" maka akan berpindah ke "`opsi2`". Proses ini berlanjut hingga mencapai sebuah "`Teka`" yang tidak memiliki opsi lebih lanjut (baik "`opsi1`" maupun "`opsi2`" adalah "`null`"), yang menandakan bahwa penyelidikan selesai.

1.5.5 *Stack Class*

```
class NodeStack {
String lokasi;
NodeStack next;

public NodeStack(String lokasi) {
this.lokasi = lokasi;
this.next = null;
}
```

```
}
class Stack {
private NodeStack top; // Titik atas stack
public Stack() {
this.top = null;
}
```

Kelas "NodeStack" merepresentasikan simpul dalam struktur data "stack", dengan atribut lokasi untuk menyimpan data dan "next" sebagai referensi ke simpul berikutnya. Konstruktor kelas ini menginisialisasi "lokasi" dengan nilai yang diberikan dan "next" ke "null". Sementara itu, kelas "Stack" mewakili "stack" itu sendiri, dengan atribut "top" yang menunjuk ke simpul teratas "stack", dan konstruktor "Stack()" menginisialisasi "top" ke "null", menandakan "stack" kosong.

```
public void push(String lokasi) {
NodeStack newNode = new NodeStack(lokasi); // node baru
newNode.next = top; // tamabah ke stack
top = newNode;
System.out.println("Anda telah mengunjungi lokasi: " + lokasi);
}
public String pop() {
if (isEmpty()) {
System.out.println("Stack kosong! Tidak ada lokasi untuk dihapus.");
return null;
}
String lokasiDihapus = top.lokasi; // ambil lokasi dari stack
top = top.next; // Pindah ke node berikutnya
System.out.println("Kembali dari lokasi: " + lokasiDihapus);
return lokasiDihapus;
}
public boolean isEmpty() {
return top == null; // cek apakah stack kosong
}
public void display() {
if (isEmpty()) {
System.out.println("Belum ada lokasi yang dikunjungi.");
} else {
System.out.println("Jejak perjalanan:");
NodeStack current = top;
while (current != null) {
System.out.println("-  "  +  current.lokasi);  //  Menampilkan  jejak
       perjalanan
current = current.next;
}
}
}
}
```

Metode-metode dalam kelas "Stack" digunakan untuk mengelola lokasi dalam "stack". Metode "push"(String lokasi) menambahkan lokasi baru ke "stack" dengan membuat simpul baru dan menempatkannya di atas "stack", lalu memperbarui "top". Metode "pop()" menghapus elemen teratas dari "stack", mengambil lokasi dari simpul

"top", dan memperbarui "top" untuk menunjuk ke simpul berikutnya. Jika "stack" kosong, pesan peringatan akan ditampilkan. Metode "isEmpty()" memeriksa apakah "stack" kosong dengan mengembalikan "true" jika "top" adalah "null". Terakhir, "display()" menampilkan semua lokasi dalam "stack".

1.5.6 Petunjuk *Class*

```
class Tunjuk {
String clue;
Tunjuk next;
public Tunjuk(String clue) {
this.clue = clue;
this.next = null;
}
}
class Petunjuk {
private String clue;
public Petunjuk() {
this.clue = "";
}
public void addClue(String clue) {
this.clue = clue;
}
public String getClue() {
return this.clue;
}
}
```

Kelas "Tunjuk" merepresentasikan simpul dalam struktur data yang menyimpan petunjuk berupa "string(clue)" dan referensi ke simpul berikutnya. Konstruktor kelas ini menginisialisasi "clue" dengan nilai yang diberikan dan "next" ke "null". Sementara itu, kelas "Petunjuk" memiliki atribut "clue" untuk menyimpan petunjuk, yang diinisialisasi dengan "string" kosong, serta metode "addClue String clue" untuk menambah atau mengubah petunjuk dan "getClue()" untuk mengambil nilai petunjuk yang disimpan.

1.5.7 *Graph Class*

```
class Node {
String name;
Petunjuk petunjuk;
Node next;
Node connections;

public Node(String name) {
this.name = name;
this.petunjuk = new Petunjuk();
this.next = null;
this.connections = null;
}
```

Kelas "Node" merepresentasikan simpul yang menyimpan nama entitas "name", objek "Petunjuk" untuk petunjuk terkait, dan referensi ke simpul berikutnya "next" serta koneksi ke simpul lain "connections". Konstruktor "Node" "String name" menginisialisasi "nama", membuat objek "Petunjuk".

```
public void addClue(String clue) {
this.petunjuk.addClue(clue);
}
public void addConnection(Node node) {
Node newConnection = new Node(node.name);
newConnection.next = this.connections;
this.connections = newConnection;
}
public void displayClue() {
if (this.petunjuk != null) {
System.out.println("Petunjuk di lokasi " + this.name + ": " +
       this.petunjuk.getClue());
}
}
public void displayConnections() {
if (this.connections == null) {
System.out.println("Tidak ada lokasi yang terhubung.");
return;
}
System.out.println("Lokasi terhubung dengan " + this.name + ":");
Node current = this.connections;
while (current != null) {
System.out.println("- " + current.name);
current = current.next;
}
}
}
```

Metode-metode dalam kelas "Node" digunakan untuk mengelola petunjuk dan koneksi antar simpul. Metode "addClue", "String clue" menambahkan petunjuk ke objek "Petunjuk" yang terkait dengan simpul saat ini. Metode "addConnection" menambah koneksi antara simpul saat ini dengan simpul lain dengan membuat simpul baru dan menghubungkannya ke daftar koneksi yang ada. Sementara itu, "displayClue()" menampilkan petunjuk yang terkait dengan simpul ini, jika ada. Terakhir, "displayConnections()" menampilkan semua simpul yang terhubung dengan simpul ini. Jika tidak ada koneksi, akan ditampilkan pesan bahwa "tidak ada lokasi yang terhubung", dan jika ada, daftar lokasi yang terhubung akan ditampilkan.

```
class Graph {
private Node head;

public void addLokasi(String name) {
Node newNode = new Node(name);
```

```
if (head == null) {
head = newNode;
} else {
Node current = head;
while (current.next != null) {
current = current.next;
}
current.next = newNode;
}
}
public Node cariLokasi(String name) {
Node current = head;
while (current != null) {
if (current.name.equals(name)) {
return current;
}
current = current.next;
}
Return;
       }
```

Kelas "`Graph`" digunakan untuk mengelola sekumpulan simpul yang mewakili lokasi-lokasi dalam sebuah graf. Metode "`addLokasi`" menambahkan lokasi baru ke dalam graf dengan membuat simpul baru "`newNode`" berdasarkan nama yang diberikan. Jika "`head`" masih "`null`", maka"`head`" akan diisi dengan simpul baru tersebut, jika tidak, simpul baru akan ditambahkan ke akhir daftar dengan mengiterasi melalui daftar hingga menemukan simpul terakhir dan menghubungkannya ke simpul baru. Metode "`cariLokasi`" mencari lokasi berdasarkan nama yang diberikan dengan mengiterasi melalui daftar dan memeriksa setiap simpul hingga lokasi ditemukan. Jika "`lokasi`" tidak ditemukan, metode ini mengembalikan "`null`".

```
public void displayLoc() {
if (head == null) {
System.out.println("Belum ada lokasi yang ditambahkan.");
return;
}
System.out.println("Daftar lokasi:");
Node current = head;
while (current != null) {
System.out.println("> " + current.name);
current = current.next;
}
}
public void sortLocations() {
if (head == null || head.next == null) {
return;
}
boolean swapped;
do {
swapped = false;
```

```
Node current = head;
while (current.next != null) {
if (current.name.compareTo(current.next.name) > 0) {
String temp = current.name;
current.name = current.next.name;
current.next.name = temp;
swapped = true;
}
current = current.next;
}
} while (swapped);
System.out.println("Lokasi berhasil diurutkan.");
}
public void connectLokasi(String loc1, String loc2) {
Node node1 = cariLokasi(loc1);
Node node2 = cariLokasi(loc2);
if (node1 != null && node2 != null) {
node1.addConnection(node2);
node2.addConnection(node1);
}
}
}
```

Metode-metode dalam kelas "`Graph`" digunakan untuk mengelola dan menampilkan lokasi-lokasi dalam *graf*. Metode "`displayLoc()`" menampilkan semua lokasi yang ada dalam *graf*, dimulai dari "`head`". Jika *graf* kosong, pesan "`Belum ada lokasi yang ditambahkan`" ditampilkan. Metode "`sortLocations()`" mengurutkan lokasi-lokasi dalam *graf* secara alfabet dengan menggunakan algoritma *bubble sort*. Jika *graf* kosong atau hanya memiliki satu lokasi, metode ini tidak melakukan apa-apa. Setelah pengurutan selesai, pesan "`Lokasi berhasil diurutkan`" ditampilkan. Terakhir, metode "`connectLokasi`" menghubungkan dua lokasi yang ada dalam *graf*. Jika kedua lokasi ditemukan, koneksi dua arah ditambahkan antara kedua simpul tersebut.

1.5.8 Ayo Main *Class*

```
import java.util.Scanner;

class AyoMain {
private int skor;
private final Graph lokasiGraph;
private final Stack jejakPerjalanan;
private final TekaTeki tekaTeki;
private static final int skor_max = 100;
private static final int penalti = 10;
private static final int skor_awal = 0;
private String tersangkaTerpilih;

public AyoMain() {
lokasiGraph = new Graph();
```

```
jejakPerjalanan = new Stack();
tekaTeki = new TekaTeki();
tersangkaTerpilih = null;
skor = skor_max;
}
public void startGame() {
initializeGame();
Scanner scanner = new Scanner(System.in);
boolean running = true;

while (running) {
System.out.println("\nSkor : " + skor );
System.out.println("╔═══════════════════════════╗");
System.out.println("║ Crime Solver Menu ║");
System.out.println("╠═══════════════════════════╣");
System.out.println("║ 1. Kunjungi Lokasi ║");
System.out.println("║ 2. Lihat Jejak Perjalanan║");
System.out.println("║ 3. Lihat Semua Lokasi ║");
System.out.println("║ 4. Selesaikan Teka-Teki ║");
System.out.println("║ 5. Pilih Tersangka ║");
System.out.println("║ 6. Selesaikan Kasus ║");
System.out.println("║ 7. Keluar ║");
System.out.println("╚═══════════════════════════╝");
System.out.print("Pilihan Anda: ");

int choice = scanner.nextInt();
scanner.nextLine();
```

Kelas "`AyoMain`" adalah inti dari permainan *Crime Solver* yang mengelola alur permainan. Dalam konstruktor, objek "`Graph`" untuk menyimpan "`Lokasi`", "`Stack`" untuk melacak jejak perjalanan, dan "`TekaTeki`" untuk teka-teki yang harus diselesaikan, semuanya diinisialisasi. "`Skor`" dimulai dari nilai maksimal "`skor_max`" dan berkurang dengan penalti setiap kali pemain membuat kesalahan. Dalam metode "`startGame()`", menu permainan ditampilkan dalam bentuk antarmuka berbasis teks.

```
switch (choice) {
case 1:
System.out.println("Pilih lokasi yang ingin dikunjungi:");
lokasiGraph.displayLoc();
System.out.print("Masukkan nama lokasi: ");
String visitLocation = scanner.nextLine();
Node lokasi = lokasiGraph.cariLokasi(visitLocation);
if (lokasi != null) {
jejakPerjalanan.push(visitLocation);
System.out.println("Kamu telah mengunjungi lokasi: " + visitLocation);
lokasi.displayClue();
lokasi.displayConnections();
mulaiPenyelidikan(lokasi);
} else {
System.out.println("Lokasi tidak ditemukan!");
skor -= penalti;
if(gameOver()){
return;
```

```
}
}
break;
case 2:
jejakPerjalanan.display();
break;
case 3:
lokasiGraph.displayLoc();
break;
case 4:
tekaTeki.solve();
break;
case 5:
pilihTersangka(scanner);
break;
case 6:
solveCase(scanner);
break;
case 7:
running = false;
System.out.println("Terima kasih sudah bermain!");
break;
default:
System.out.println("Pilihan tidak valid. Coba lagi.");
}
}
}
private void mulaiPenyelidikan(Node lokasi) {
Scanner scanner = new Scanner(System.in);
boolean selesai = false;
boolean petunjukSudahDiperiksa = false;
while (!selesai) {
System.out.println("\n=== Penyelidikan di " + lokasi.name + " ===");
System.out.println("1. Periksa petunjuk lebih detail");
System.out.println("2. Kembali ke Menu Utama");
//System.out.println("3. Kembali ke Menu Utama");
System.out.print("Pilihan Anda: ");
int pilihan = scanner.nextInt();
scanner.nextLine();
switch (pilihan) {
case 1:
if (petunjukSudahDiperiksa) {
System.out.println("Petunjuk tambahan ditemukan!");
tampilkanPetunjukTambahan(lokasi);
} else {
System.out.println("Memeriksa petunjuk lebih detail...");
lokasi.displayClue();
petunjukSudahDiperiksa = true;
}
break;
case 2:
selesai = true;
break;
default:
System.out.println("Pilihan tidak valid.");
}
}
```

```
}
```

Bagian kode ini menangani interaksi pemain dengan menu permainan *Crime Solver*. Dalam metode "`startGame()`", berbagai pilihan disediakan kepada pemain, termasuk "`mengunjungi lokasi`", "`melihat jejak perjalanan`", "`menampilkan lokasi`", "`menyelesaikan teka-teki`", "`memilih tersangka`", atau "`menyelesaikan kasus`". Pemain dapat memilih lokasi untuk dikunjungi, dan jika lokasi ditemukan, program mencatatnya dalam jejak perjalanan, menampilkan petunjuk, serta menghubungkan lokasi dengan lokasi lain. Pemain kemudian dapat melanjutkan penyelidikan di lokasi yang dikunjungi, memeriksa petunjuk lebih detail atau kembali ke menu utama. Penalti diberikan jika lokasi yang dipilih tidak ditemukan, dan permainan akan berakhir jika kondisi "`gameover`" tercapai. Metode "`mulaiPenyelidikan`" memungkinkan pemain untuk memeriksa petunjuk lebih mendalam di lokasi yang dipilih, dengan hanya satu kesempatan untuk memeriksa petunjuk awal, sebelum petunjuk tambahan ditampilkan jika ditemukan.

```
private void tampilkanPetunjukTambahan(Node lokasi) {
System.out.println("Petunjuk baru yang ditemukan: ");
if (lokasi.name.equals("TKP")) {
System.out.println("Tersangka A memiliki bekas luka di tangannya, yang
       mungkin terkait dengan noda darah di TKP.");
} else if (lokasi.name.equals("Kamar Tersangka")) {
System.out.println("Ada surat yang tergeletak di meja di kamar
       tersangka. Mungkin ini terkait dengan motif kejahatan.");
} else if (lokasi.name.equals("Dapur")) {
System.out.println("Tersangka A tampaknya sering berada di dapur. Ada
       noda yang belum kering di lantai.");
} else if (lokasi.name.equals("Taman")) {
System.out.println("Beberapa tanaman tampaknya baru dipindahkan, bisa
       jadi untuk menyembunyikan jejak.");
} else if (lokasi.name.equals("Garasi")) {
System.out.println("Ada bekas oli tumpah di lantai garasi. Mungkin
       mobil tersangka A digunakan untuk melarikan diri.");
} else if (lokasi.name.equals("Gudang")) {
System.out.println("Di gudang, ada petunjuk tambahan berupa barang
       yang terbalik. Mungkin ada sesuatu yang disembunyikan.");
}
}
```

Metode"`tampilkanPetunjukTambahan`" digunakan untuk menampilkan petunjuk tambahan yang ditemukan di lokasi tertentu berdasarkan nama lokasi yang diberikan. Setiap lokasi memiliki petunjuk spesifik yang dapat membantu pemain dalam memecahkan kasus. Misalnya, jika lokasi adalah "`TKP`", petunjuk yang ditemukan adalah bekas luka di tangan tersangka yang mungkin terkait dengan noda darah di

tempat kejadian. Di lokasi lain seperti "Kamar `Tersangka`", terdapat surat yang bisa menjadi motif kejahatan, sementara di "`Dapur`", ada noda yang mungkin menunjukkan aktivitas mencurigakan tersangka. Lokasi lainnya seperti "`Taman`", "`Garasi`", dan "`Gudang`" masing-masing memberikan petunjuk yang dapat membantu dalam penyelidikan lebih lanjut, seperti jejak yang disembunyikan atau barang yang terbalik, yang menunjukkan ada yang disembunyikan.

```
private void initializeGame() {
lokasiGraph.addLokasi("TKP");
lokasiGraph.addLokasi("Kamar Tersangka");
lokasiGraph.addLokasi("Dapur");
lokasiGraph.addLokasi("Taman");
lokasiGraph.addLokasi("Garasi");
lokasiGraph.addLokasi("Gudang");

Node tKPLokasi = lokasiGraph.cariLokasi("TKP");
if (tKPLokasi != null) {
tKPLokasi.addClue("Sidik jari ditemukan di meja. Ada noda darah di
      lantai. Pelaku terindikasi A.");
}
Node kamarTersangka = lokasiGraph.cariLokasi("Kamar Tersangka");
if (kamarTersangka != null) {
kamarTersangka.addClue("Hasil DNA cocok dengan tersangka A. Ada jejak
      sepatu di karpet.");
}
Node dapur = lokasiGraph.cariLokasi("Dapur");
if (dapur != null) {
dapur.addClue("Saksi mendengar suara keras dari dapur pada malam
      kejadian.");
}
Node taman = lokasiGraph.cariLokasi("Taman");
if (taman != null) {
taman.addClue("Tersangka A sering terlihat di taman. Terdapat surat
      yang sobek.");
}
Node garasi = lokasiGraph.cariLokasi("Garasi");
if (garasi != null) {
garasi.addClue("Ada bekas ban mobil di tanah. Mobil tersangka A
      terparkir di sana.");
}
Node gudang = lokasiGraph.cariLokasi("Gudang");
if (gudang != null) {
gudang.addClue("Ada jejak kaki basah yang mengarah ke gudang.
      Ditemukan barang bukti di sana.");
}
lokasiGraph.connectLokasi("TKP", "Kamar Tersangka");
lokasiGraph.connectLokasi("TKP", "Dapur");
lokasiGraph.connectLokasi("TKP", "Taman");
lokasiGraph.connectLokasi("Kamar Tersangka", "Garasi");
lokasiGraph.connectLokasi("Taman", "Gudang");
lokasiGraph.connectLokasi("Dapur", "Gudang");
}
```

Metode "`initializeGame()`" berfungsi untuk menginisialisasi permainan dengan menambahkan berbagai lokasi ke dalam graf, yang masing-masing memiliki petunjuk terkait. Lokasi-lokasi tersebut antara lain "`TKP`", "`Kamar Tersangka`", "`Dapur`", "`Taman`", "`Garasi`", dan "`Gudang`". Setiap lokasi dihubungkan dengan petunjuk yang relevan, seperti sidik jari yang ditemukan di TKP atau DNA yang cocok dengan tersangka A di kamar tersangka. Lokasi-lokasi ini juga dihubungkan satu sama lain untuk menciptakan peta perjalanan yang dapat dieksplorasi pemain, seperti menghubungkan "`TKP`" dengan "`kamar tersangka`", "`dapur`", dan "`taman`", serta lokasi lainnya seperti "`garasi`" dan "`gudang`". Tujuan dari inisialisasi ini adalah untuk menyediakan jalur dan informasi yang akan digunakan pemain dalam penyelidikan kasus.

```
private void solveCase(Scanner scanner) {
System.out.println("\n=== Menyelesaikan Kasus ===");
if (tersangkaTerpilih == null) {
System.out.println("Tersangka belum dipilih. Silakan kunjungi lebih
      banyak lokasi untuk mendapatkan petunjuk.");
skor -= penalti;
if(gameOver()){
return;
}
return;
}
System.out.println("Berdasarkan investigasi, sepertinya kamu
      mencurigai " + tersangkaTerpilih + " sebagai tersangka");

String alibi = "";
if (tersangkaTerpilih.equals("Tersangka A")) {
alibi = "Tersangka A mengaku berada di rumah pada malam kejadian dan
      tidak meninggalkan rumah sama sekali.";
} else if (tersangkaTerpilih.equals("Tersangka B")) {
alibi = "Tersangka B mengaku berada di luar kota saat kejadian, dan
      ada bukti bahwa dia benar-benar berada di tempat lain.";
} else if (tersangkaTerpilih.equals("Tersangka C")) {
alibi = "Tersangka C mengaku tidur di kamar tidurnya pada malam
      kejadian, yang dapat dibuktikan oleh keluarganya.";
}

System.out.println("Alibi yang diberikan oleh tersangka: " + alibi);
System.out.println("Apakah kamu yakin dia tersangkanya? (y/n): ");
String pilihan = scanner.nextLine().toLowerCase();
if(pilihan.equals("y")){
if(tersangkaTerpilih.equals("Tersanka A")){
System.out.println("Kamu benar pelakunya adalah tersangka A");
skor += 20;
}else{
System.out.println("Kamu menuduh orang yang tidak bersalah.");
skor -= penalti;
```

```
if(gameOver()){
return;
}
}
}else{
System.out.println("Kamu menyerah pada penyilidikan ini");
skor -= penalti;
}
System.out.println("Kasus sudah berakhir, skor akhirmu: " + skor);
}
```

Metode "`solveCase()`" digunakan untuk menyelesaikan kasus setelah pemain melakukan penyelidikan. Jika tersangka belum dipilih, pemain diminta untuk mengunjungi lebih banyak lokasi untuk mengumpulkan petunjuk, dan skor akan berkurang sebagai penalti. Setelah tersangka dipilih, pemain diberikan informasi terkait alibi tersangka, seperti keterangan yang diberikan oleh masing-masing tersangka mengenai keberadaan mereka pada malam kejadian. Pemain kemudian diminta untuk memutuskan apakah mereka yakin dengan pilihan tersangka atau tidak. Jika pemain memilih "`y`" dan benar memilih tersangka yang tepat, skor akan bertambah, namun jika salah, skor akan berkurang. Jika pemain menyerah, penalti juga akan diterapkan. Setelah kasus diselesaikan, skor akhir ditampilkan.

```
private void pilihTersangka(Scanner scanner){
System.out.println("\nSilahkan pilih orang yang kamu curigai");
System.out.println("1. Tersangka A");
System.out.println("2. Tersangka B");
System.out.println("3. Tersangka C");

System.out.print("Pilih nomor tersangka: ");
int pilihan = scanner.nextInt();
scanner.nextLine();
switch (pilihan) {
case 1:
tersangkaTerpilih = "Tersangka A";
break;
case 2:
tersangkaTerpilih = "Tersangka B";
break;
case 3:
tersangkaTerpilih = "Tersangka C";
break;
default:
System.out.println("Pilihan tidak valid!");
return;
}
System.out.println("Kamu sepertinya mencurigai " + tersangkaTerpilih
       + " sebagai pelaku.");

String alibi = "";
if (tersangkaTerpilih.equals("Tersangka A")) {
```

```
alibi = "Tersangka A mengaku berada di rumah pada malam kejadian dan
       tidak meninggalkan rumah sama sekali.";
} else if (tersangkaTerpilih.equals("Tersangka B")) {
alibi = "Tersangka B mengaku berada di luar kota saat kejadian, dan
       ada bukti bahwa dia benar-benar berada di tempat lain.";
} else if (tersangkaTerpilih.equals("Tersangka C")) {
alibi = "Tersangka C mengaku tidur di kamar tidurnya pada malam
       kejadian, yang dapat dibuktikan oleh keluarganya.";
}

System.out.println("Alibi " + tersangkaTerpilih + ": " + alibi);
}
private boolean gameOver(){
if(skor <= skor_awal){
System.err.println("\nGAME OVER!!!");
System.out.println("Kamu kalah! Kasus tidak terpecahkan");
return true;
}
return false;
}
}
```

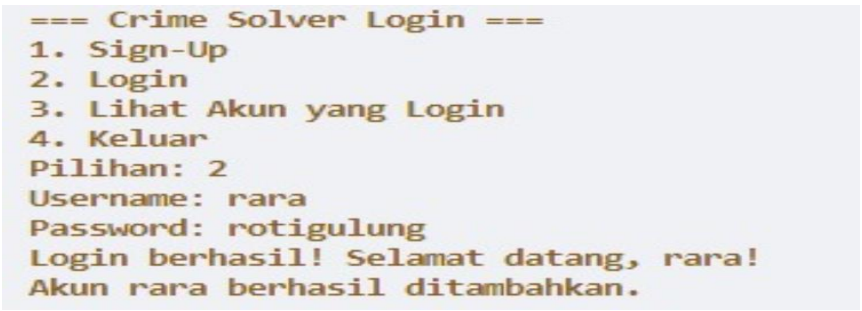
Metode "`pilihTersangka()`" memungkinkan pemain untuk memilih tersangka yang mereka curigai berdasarkan pilihan yang tersedia. Pemain diminta untuk memilih salah satu dari tiga tersangka dan setelah memilih, sistem menampilkan alibi tersangka yang dipilih. Alibi ini mencakup keterangan mengenai keberadaan tersangka pada malam kejadian, yang bisa mempengaruhi keputusan pemain dalam menyelesaikan kasus. Jika pemain memilih pilihan yang tidak *valid*, pesan kesalahan akan ditampilkan. Selain itu, metode "`gameOver()`" digunakan untuk memeriksa apakah permainan sudah berakhir, yakni jika skor pemain sudah mencapai nilai awal atau lebih rendah, yang berarti pemain gagal memecahkan kasus dan kalah dalam permainan.

1.6 *Output* Program

Berikut merupakan *output* dari program Struktur Data pada game Crime Solver yang telah kami buat:

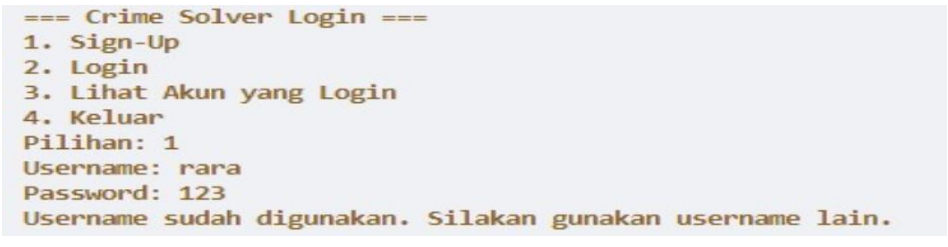
Gambar 1.1 *sign-up* dan *login*

Berdasarkan **Gambar 1.1** dapat diketahui bahwa gambar tersebut merupakan output yang dihasilkan dari program kami. Pada tampilan ini, *user* diharuskan untuk melakukan proses *sign-up* dengan mengisi *username* dan *password*. Jika user sudah berhasil mendaftar, maka *user* dapat melakukan *login* untuk memulai permainan.



Gambar 1.2 *Login*

Berdasarkan **Gambar 1.2** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* berhasil melakukan proses *login*. Proses ini dilakukan dengan memasukkan *username* dan *password* yang telah dibuat sebelumnya pada proses *sign-up.*



Gambar 1.3 *Login error*

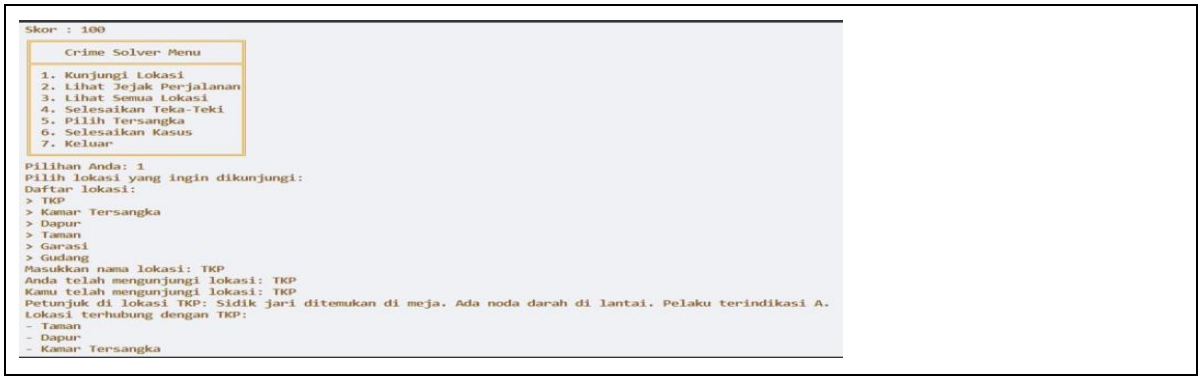
Berdasarkan **Gambar 1.3** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* mencoba melakukan proses *sign-up* dengan *username* dan *password* yang sudah digunakan sebelumnya.

Gambar 1.4 Menu Utama

Berdasarkan **Gambar 1.4** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* telah berhasil memasuki menu utama permainan. Pada menu ini, akan ditampilkan *score* awal sebagai nilai dalam permainan, dan juga beberapa opsi seperti:

1. Kunjungi Lokasi, 2. Lihat Jejak Perjalanan, 3. Lihat Semua Lokasi, 4. Selesaikan Teka-Teki, 5. Pilih Tersangka, 6. Selesaikan Kasus, dan 7. Keluar.

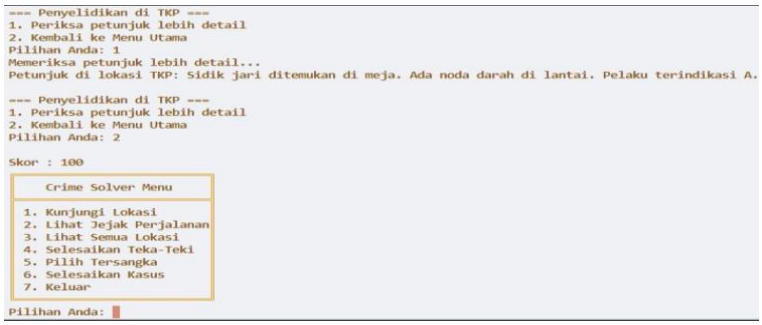
Pada gambar tersebut, *user* memilih opsi "Kunjungi Lokasi" yang akan menampilkan daftar lokasi yang dapat dikunjungi untuk mencari petunjuk dalam permainan.



Gambar 1.5 Tampilan Kunjungi Lokasi

Berdasarkan **Gambar 1.5** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih lokasi "TKP". Lokasi ini memberikan informasi seperti "Sidik jari ditemukan di meja. Ada noda darah di lantai. Pelaku terindikasi A". Selain itu *user* juga diberitahu bahwa terdapat beberapa lokasi lain yang terhubung dengan TKP,

seperti taman, dapur dan kamar tersangka. Petunjuk ini bertujuan untuk membantu *user* dalam menyelesaikan permainan.



Gambar 1.6 Tampilan Investigasi

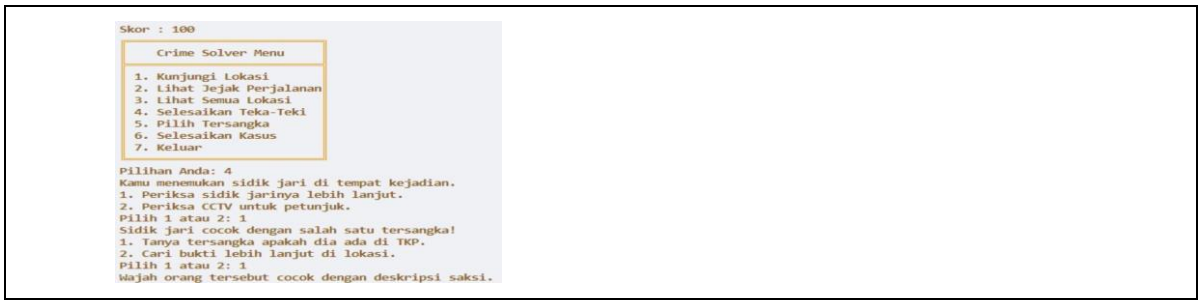
Berdasarkan **Gambar 1.6** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* telah menyelesaikan proses pada **Gambar 1.5**. Pada tahap ini, ditampilkan dua opsi yaitu: 1. Periksa Petunjuk lebih detail dan 2. Kembali ke Menu Utama. Pada gambar tersebut, *user* memilih opsi kedua, yang akan menampilkan kembali tampilan menu utama permainan.

Gambar 1.7 Tampilan Jejak Perjalanan

Berdasarkan **Gambar 1.7** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi "Lihat Jejak Perjalanan". Opsi ini menampilkan riwayat perjjalanan yang telah dilakukan oleh *user* selama permainan berlangsung.

Gambar 1.8 Tampilan Lokasi

Berdasarkan **Gambar 1.8** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi "Lihat Semua Lokasi". Opsi ini menampilkan beberapa daftar lokasi yang menjadi petunjuk dalam permainan.



Gambar 1.9 Tampilan Menyelesaikan Teka Teki

Berdasarkan **Gambar 1.9** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi "Selesaikan Teka-Teki". Opsi ini menampilkan informasi berupa "Kami menemukan sidik jari di tempat kejadian". *User* kemudian diberikan dua pilihan untuk menyelidiki lebih lanjut, yaitu, "Periksa sidik jari lebih lanjut" dan "Periksa CCTV untuk petunjuk". Pada gambar tersebut, *user* memilih opsi pertama yang menampilkan informasi berupa "Sidik jari cocok dengan salah satu tersangka". Setelah itu, muncul dua pilihan baru, yaitu, "Tanya tersangka apakah dia ada di TKP" dan "Cari bukti lebih lanjut di lokasi". *User* memilih opsi pertama, yang memberikan informasi bahwa, "Wajah orang tersebut cocok dengan deskripsi".

Gambar 1.10 Tampilan Mencurigai Pelaku

Berdasarkan **Gambar 1.10** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi "Pilih Tersangka". Opsi ini menampilkan beberapa pilihan tersangka, seperti: 1.Tersangka A, 2. Tersangka B, 3. Tersangka C.

Pada gambar tersebut, *user* memilih opsi kedua yaitu "Tersangka B". Pilihan ini menampilkan informasi detail yang berisi, "Tersangka B mengaku berada di luar kota saat kejadian, dan ada bukti bahwa dia benar-benar berada di tempat lain".

Gambar 1.11 Tampilan Memilih Tersangka Yang Salah

Berdasarkan **Gambar 1.11** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi, "Selesaikan Kasus". Opsi ini menampilkan pertanyaan kepada *user* yang berisi "Apakah kamu yakin dia tersangkanya?", dengan dua pilihan jawaban, yaitu "y" (ya) dan "n" (tidak).

Pada gambar tersebut, *user* memilih opsi "y". Jika jawaban yang dipilih ternyata salah, maka akan muncul informasi berupa "Kamu menuduh orang yang tidak bersalah“. Selain itu, *score user* akan berkurang sebagai konsekuensi dari jawaban yang salah.

Gambar 1.12 Tampilan Memilih Tersangka

Berdasarkan **Gambar 1.12** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi "Pilih Tersangka". Opsi ini menampilkan beberapa pilihan tersangka, seperti: 1.Tersangka A, 2. Tersangka B, 3. Tersangka C.

Pada gambar tersebut, *user* memilih opsi pertama yaitu "Tersangka A". Pilihan ini menampilkan informasi detail yang berisi, "Tersangka A mengaku berada di rumah pada malam kejadian dan tidak meninggalkan rumah sama sekali".

Gambar 1.13 Tampilan Menyelesaikan Permainan

Berdasarkan **Gambar 1.13** dapat diketahui bahwa gambar tersebut merupakan output yang menunjukkan hasil ketika *user* memilih opsi, "Selesaikan Kasus". Opsi ini menampilkan pertanyaan kepada *user* yang berisi "Apakah kamu yakin dia tersangkanya?", dengan dua pilihan jawaban, yaitu "y" (ya) dan "n" (tidak).

Pada gambar tersebut, *user* memilih opsi "y". Jika jawaban yang dipilih ternyata salah, maka akan muncul informasi berupa "Kamu menuduh orang yang tidak bersalah". Selain itu, *score user* akan berkurang sebagai konsekuensi dari jawaban yang salah. Namun, jika jawaban yang dipilih benar, permainan akan berakhir dan menampilkan skor akhir dari permainan.

1.7 Kesimpulan

Berdasarkan hasil *project* praktikum Algoritma dan Struktur Data, dapat disimpulkan bahwa permainan "Crime Solver: Petualangan Detektif Misteri di Balik Lokasi" yang diimplementasikan dalam bahasa pemrograman dapat menunjukkan suatu upaya dalam membangun sistem permainan yang terstruktur dan efisien. Dalam permainan ini *linked list* digunakan sebagai struktur data dasar untuk menyimpan daftar akun yang mendaftar dalam permainan. Selanjutnya, implementasi struktur data *graph* untuk hubungan antar lokasi yang dapat terhubung dengan lokasi lain. Selain itu, konsep *stack* dan *queue* digunakan untuk menyimpan jejak perjalanan pemain dan menyimpan akun-akun pemain yang berhasil login. Pencarian akun atau lokasi didalam permainan diimplementasikan dengan menggunakan *method* pencarian linear. Keseluruhan, implementasi ini mencakup berbagai konsep dan struktur data, menciptakan sistem permainan yang efisien dalam penelusuran data dan pengambilan keputusan berbasis petunjuk.

1.8 Referensi

[1] A Soetedjo, R Sidik - Jurnal Teknologi dan Informasi, 2019

[2] H Saputra, TH Kusmanto Seminar Nasional Riset, 2023

[3] H Hurnaningsih - Journal of Research and Publication …, 2023