Laporan Tugas Kecil 3
IF2211 Strategi Algoritma
Penyelesaian Puzzle Rush Hour
Menggunakan Algoritma Pathfinding

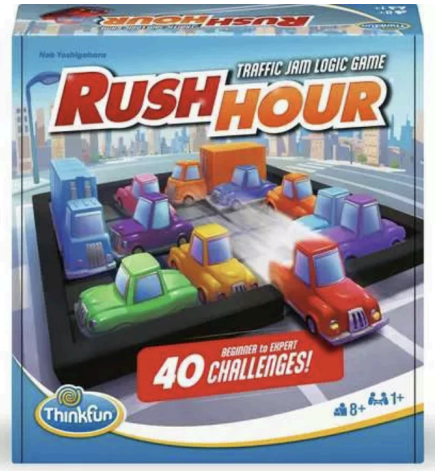
Disusun Oleh
13523021 - Muhammad Raihan Nazhim Oktana
13523095 - Rafif Farras

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH DAN ALGORITMA

1.1. Deskripsi Masalah



Gambar 1. Rush Hour Puzzle
(Sumber: https://www.thinkfun.com/en-US/products/educational-games/rush-hour-76582)

Rush Hour adalah sebuah permainan puzzle logika berbasis grid yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6x6) agar mobil utama (biasanya berwarna merah) dapat keluar dari kemacetan melalui pintu keluar di sisi papan. Setiap kendaraan hanya bisa bergerak lurus ke depan atau ke belakang sesuai dengan orientasinya (horizontal atau vertikal), dan tidak dapat berputar. Tujuan utama dari permainan ini adalah memindahkan mobil merah ke pintu keluar dengan jumlah langkah seminimal mungkin.

Komponen penting dari permainan Rush Hour terdiri dari:

1. **Papan** – *Papan* merupakan tempat permainan dimainkan.
   Papan terdiri atas *cell*, yaitu sebuah *singular point* dari papan. Sebuah *piece* akan menempati *cell-cell* pada papan. Ketika permainan dimulai, semua *piece* telah diletakkan di dalam papan dengan konfigurasi tertentu berupa lokasi piece dan *orientasi*, antara *horizontal* atau *vertikal*.
   Hanya *primary piece* yang dapat digerakkan **keluar papan melewati *pintu keluar***. *Piece* yang bukan *primary piece* tidak dapat digerakkan keluar papan. Papan memiliki satu *pintu keluar* yang pasti berada di *dinding papan* dan sejajar dengan orientasi *primary piece*.

2. **Piece –** *Piece* adalah sebuah kendaraan di dalam papan. Setiap *piece* memiliki *posisi*, *ukuran*, dan *orientasi*. *Orientasi* sebuah *piece* hanya dapat berupa horizontal atau vertikal–tidak mungkin diagonal. *Piece* dapat memiliki beragam *ukuran*, yaitu jumlah *cell* yang ditempati oleh *piece*. Secara standar, variasi *ukuran* sebuah *piece* adalah *2-piece* (menempati 2 *cell*) atau *3-piece* (menempati 3 *cell*). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

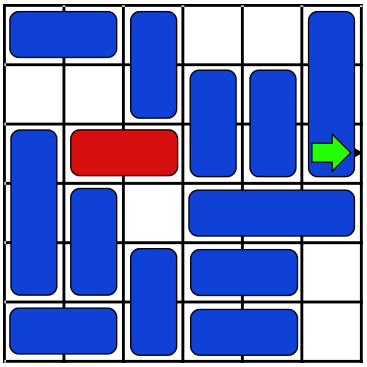
3. **Primary Piece –** *Primary piece* adalah kendaraan utama yang harus dikeluarkan dari *papan* (biasanya berwarna merah). Hanya boleh terdapat satu primary piece.

4. **Pintu Keluar –** *Pintu keluar* adalah tempat *primary piece* dapat digerakkan keluar untuk menyelesaikan permainan

5. **Gerakan** — *Gerakan* yang dimaksudkan adalah pergeseran *piece* di dalam permainan. *Piece* hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

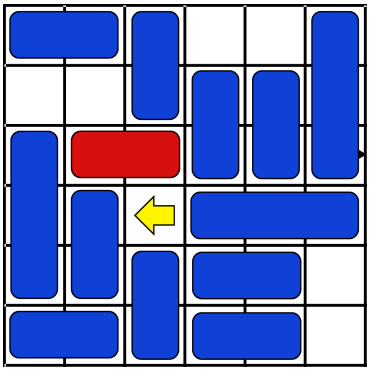
Ilustrasi kasus :

Diberikan sebuah *papan* berukuran 6 x 6 dengan 12 *piece* kendaraan dengan 1 *piece* merupakan *primary piece*. *Piece* ditempatkan pada *papan* dengan posisi dan orientasi sebagai berikut.

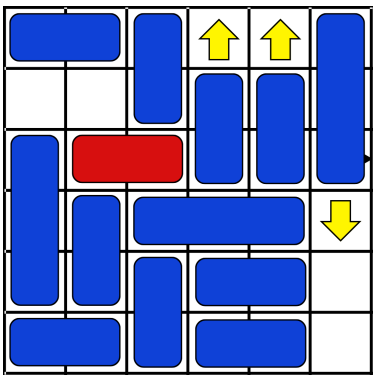


Gambar 2. Awal Permainan Game Rush Hour

Pemain dapat menggeser-geser *piece* (termasuk *primary piece*) untuk membentuk jalan lurus antara *primary piece* dan *pintu keluar*.

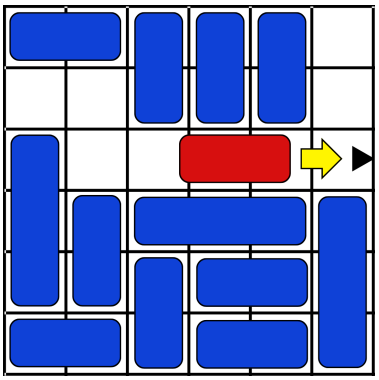


Gambar 3. Gerakan Pertama Game Rush Hour



Gambar 4. Gerakan Kedua Game Rush Hour

Puzzle berikut dinyatakan telah selesai apabila *primary piece* dapat digeser keluar papan melalui *pintu keluar*.



Gambar 5. Pemain Menyelesaikan Permainan

1.2. Algoritma Uniform Cost Search (UCS) pada Pencarian Solusi Puzzle Rush Hour

Algoritma UCS merupakan varian dari Dijkstra yang digunakan untuk mencari solusi dengan biaya total minimum. Dalam implementasi ini, digunakan sebuah priority queue yang mengurutkan node berdasarkan g(n), yaitu biaya dari start hingga node saat ini. Setiap node yang dieksekusi akan diperiksa apakah sudah mencapai kondisi goal. Jika belum, semua kemungkinan gerakan berikutnya akan dibuat dan hanya node yang belum pernah dikunjungi yang ditambahkan ke antrian. Karena UCS selalu memilih node dengan biaya kumulatif terkecil, UCS menjamin menemukan solusi optimal ketika semua biaya aksi sama (Dalam kasus permainan Rush Hour biaya setiap aksi sama). Fungsi boardKey digunakan untuk menghindari siklus dengan cara mendeteksi state yang sudah dikunjungi.

```
procedure SolveUCS(initialState: DataStructure, num: Integer) → Solution
{ Menyelesaikan puzzle menggunakan Uniform Cost Search }

Deklarasi:
    pq: PriorityQueue of Node (berdasarkan nilai g)
    visited: Set of String
    cnt: Integer
    cur, child: Node

Algoritma:
    pq ← kosong
    visited ← kosong
```

```
    cnt ← 0
    Tambahkan node awal ke pq dengan g ← 0
    Tambahkan representasi board awal ke visited

    while pq tidak kosong do
        cnt ← cnt + 1
        cur ← pq.pop()
        if cur.state adalah goal then
            return solusi berdasarkan cur
        end if

        for setiap move valid dari cur.state do
            nxt ← hasil applyMove(cur.state, move)
            key ← representasi board dari nxt
            if key belum ada di visited then
                Tambahkan key ke visited
                Tambahkan node baru ke pq dengan g ← cur.g + 1
            end if
        end for
    end while

    return null
end procedure
```

1.3. Algoritma Greedy Best First Search (GBFS) pada Pencarian Solusi Puzzle Rush Hour

Greedy Best-First Search merupakan suatu algoritma pathfinding yang berfokus pada estimasi jarak ke goal tanpa memperhatikan biaya yang sudah ditempuh. Dalam kode ini, node dimasukkan ke dalam priority queue berdasarkan h(n), yaitu nilai heuristik dari posisi saat ini ke tujuan, yang dihitung menggunakan fungsi solveHeuristic. Setiap kali sebuah node dipilih, ia akan diperiksa apakah sudah menyelesaikan puzzle. Jika belum, semua kemungkinan gerakan dihasilkan dan ditambahkan ke queue tanpa melihat apakah sudah pernah dikunjungi, gerakan yang sudah dilakukan akan di cek saat iterasi berikutnya. Karena hanya fokus ke heuristik, algoritma ini tidak selalu menghasilkan solusi optimal.

```
procedure SolveGBFS(initialState: DataStructure, num: Integer) → Solution
{ Menyelesaikan puzzle menggunakan Greedy Best-First Search }

Deklarasi:
    pq: PriorityQueue of Node (berdasarkan nilai h)
    visited: Set of String
    cnt: Integer
    cur, child: Node

Algoritma:
    pq ← kosong
    visited ← kosong
    cnt ← 0
    Tambahkan node awal ke pq dengan h ← heuristik dari initialState

    while pq tidak kosong do
        cur ← pq.pop()
        if representasi cur.state belum ada di visited then
            cnt ← cnt + 1
```

```
            Tambahkan cur.state ke visited

            if cur.state adalah goal then
                return solusi berdasarkan cur
            end if

            for setiap move valid dari cur.state do
                nxt ← hasil applyMove(cur.state, move)
                Tambahkan node baru ke pq dengan h ← heuristik dari nxt
            end for
        end if
    end while

    return null
end procedure
```

1.4. Algoritma A* pada Pencarian Solusi Puzzle Rush Hour

A* adalah algoritma pencarian yang menyeimbangkan antara biaya g(n) dan h(n). Node dipilih berdasarkan nilai total f(n) = g(n) + h(n). Jika nilai f(n) sama, akan dilihat dari h(n) yang lebih kecil untuk didahulukan. Dalam implementasi ini, digunakan struktur bestCost untuk mencatat biaya minimum yang pernah dicapai untuk setiap state, agar tidak membuang waktu untuk mengeksplorasi jalur yang lebih mahal. Selain itu, meskipun suatu node pernah dikunjungi, ia tetap akan dipertimbangkan lagi jika ditemukan jalur yang lebih murah. A* sangat cocok digunakan untuk masalah seperti Rush Hour karena dapat meminimalkan jumlah langkah dengan tetap menjamin solusi optimal, selama heuristik yang digunakan bersifat admissible.

```
procedure SolveAStar(initialState: DataStructure, num: Integer) → Solution
{ Menyelesaikan puzzle menggunakan A-Star Search }

Deklarasi:
    pq: PriorityQueue of Node (berdasarkan nilai f = g + h)
    visited: Set of String
    bestCost: Map dari String ke Integer
    cnt: Integer
    cur, child: Node
    res: Node

Algoritma:
    pq ← kosong
    visited ← kosong
    bestCost ← kosong
    cnt ← 0
    Tambahkan node awal ke pq dengan g ← 0, h ← heuristik dari initialState
    bestCost[initialKey] ← 0

    while pq tidak kosong do
        cur ← pq.pop()
        key ← representasi board dari cur.state

        if key belum di visited atau cur.g < bestCost[key] then
            cnt ← cnt + 1
            Tambahkan key ke visited
```

```
            if cur.state adalah goal then
                Jika res belum ada atau cur.g lebih kecil, simpan sebagai
res
            end if

            for setiap move valid dari cur.state do
                nxt ← hasil applyMove(cur.state, move)
                nxtKey ← representasi board dari nxt
                g ← cur.g + 1

                if nxtKey belum di visited atau g < bestCost[nxtKey] then
                    h ← heuristik dari nxt
                    bestCost[nxtKey] ← g
                    Tambahkan node baru ke pq dengan g, h, dan parent cur
                end if
            end for
        end if
    end while

    return solusi berdasarkan res jika ditemukan, atau fallback
end procedure
```

BAB 2
Analisis Algoritma

Dalam konteks algoritma pathfinding, f(n) adalah fungsi evaluasi total dari sebuah node n, yang didefinisikan sebagai f(n) = g(n) + h(n). Di sini, g(n) adalah biaya dari node awal hingga mencapai node n, sedangkan h(n) adalah estimasi biaya dari n menuju goal state atau biasa disebut juga fungsi heuristik. Jika f(n) = g(n), maka itu merupakan kasus untuk algoritma UCS (tanpa heuristik). Jika f(n) = h(n), maka itu merupakan kasus untuk algoritma GBFS. Jika f(n) = g(n) + h(n), maka itu merupakan kasus untuk algoritma A*. Jadi, secara garis besar g(n) mencerminkan seberapa jauh kita sudah melangkah, sedangkan f(n) total biaya yang bisa berasal dari g(n)/h(n)/keduanya yang digunakan untuk memprioritaskan node.

Heuristik pada algoritma A* dikatakan admissible jika estimasi biaya dari node n ke h(n) tidak overestimate/ melebihi biaya sebenarnya (h*(n)). Artinya, untuk setiap node n, nilai heuristik harus memenuhi syarat $h(n) \leq h^*(n)$. Suatu heuristik yang admissible akan menyebabkan algoritma A* yang menggunakannya selalu menghasilkan solusi optimal, sebab tidak akan terpengaruh oleh estimasi biaya yang terlalu rendah dan tetap mempertimbangkan kemungkinan jalur terbaik secara menyeluruh.

Pada penyelesaian game Rush Hour, algoritma Uniform Cost Search (UCS) bisa dikatakan sama seperti Breadth-First Search (BFS), karena pada kasus ini semua langkah memiliki biaya yang sama. Dalam kondisi tersebut, baik UCS maupun BFS akan membangkitkan node berdasarkan urutan kedalaman atau jarak dari start, sehingga urutan node yang dieksekusi dan jalur yang dihasilkan pun akan sama. Perbedaannya baru muncul ketika ada variasi biaya antar aksi.

Secara teoritis, A* akan lebih efisien daripada UCS jika menggunakan heuristik yang admissible dan tepat. UCS menjelajahi node berdasarkan total biaya tanpa mempertimbangkan arah atau jarak ke tujuan, yang dapat menyebabkan eksplorasi lebih banyak node yang sebenarnya tidak mengarah pada solusi. Sedangkan, A* menggunakan biaya sebenarnya dan memperkirakan jalan ke tujuan, sehingga bisa fokus pada jalur-jalur yang lebih memang potensial untuk mencapai solusi optimal lebih cepat.

Greedy Best First Search tidak menjamin solusi optimal karena algoritma ini hanya mempertimbangkan nilai heuristik h(n) tanpa memperhitungkan g(n). Pendekatan ini rentan terhadap penggunaan heuristik yang keliru, sehingga dapat memilih jalur yang terliihat cepat padahal punya biaya yang mahal, dan akibatnya menghasilkan solusi yang tidak optimal.

BAB 3
IMPLEMENTASI PROGRAM

3.1. File UCS.java

Method	Tipe	Deskripsi
solveUCS(DataStructure dataStructure, int num)	Solution	Method yang menyelesaikan puzzle dengan metode UCS

3.2. File GBFS.java

Method	Tipe	Deskripsi
solveGBFS(DataStructure dataStructure, int num)	Solution	Method yang menyelesaikan puzzle dengan metode Greedy Best First Search

3.3. File AStar.java

Method	Tipe	Deskripsi
solveAStar(DataStructure dataStructure, int num)	Solution	Method yang menyelesaikan puzzle dengan metode A*

3.4. File Heuristic.java

Method	Tipe	Deskripsi
solveHeuristic(DataStructure dataStructure , int num)	int	Menghitung nilai heuristik dari dataStructure dan tipe heuristiknya.
heuristic1(DataStructure dataStructure)	int	Menghitung jumlah piece yang menghalangi primary piece.
heuristic2(DataStructure dataStructure)	int	Menghitung jumlah piece yang menghalangi primary piece dan jarak kosong yang tersedia.
heuristic3(DataStructure dataStructure)	int	Menghitung jumlah piece yang menghalangi primary piece dan ukuran setiap piece yang menghalangi tersebut.
blockingCount(DataStructure dataStructure)	int	Menghitung jumlah block yang menghalangi P ke K
findPrimary(DataStructure dataStructure)	Piece	Mencari primary piece dari dataStructure.
isBlocking(Piece otherPiece , Piece primaryPiece)	Boolean	Mengecek apakah piece lain menghalangi primary piece.

3.5. File Solution.java

Method	Tipe	Deskripsi
buildSolution(String algorithm , int heuristicId , int visitedCount , double time , Node goal)	Solution	Membangun tipe data Solution dari hasil algoritma pencarian
displaySolution()	void	Menampilkan solusi dalam format CLI yang terstruktur
displayColoredBoard(DataStructure data, DataStructure prevData, GameLogic.Move move)	void	Display Board From Data Structure To CLI Terminal dengan warna

3.6. File GameLogic.java

Method	Tipe	Deskripsi
boardKey(DataStructure dataStructure)	String	Membangun string linear dari dataStructure untuk digunakan sebagai key pada HashSet & HashMap
generateMoves(DataStructure dataStructure)	List<Move>	Melakukan generate semua kemungkinan gerakan piece.
applyMove(DataStructure dataStructure , Move move)	DataStructure	Melakukan suatu gerakan dan mengupdate board permainan.
scanHorizontal(Piece piece , char[][] grid , Direction direction)	List<Move>	Melakukan scanning kemungkinan move secara horizontal.
scanVertikal(Piece piece , char[][] grid , Direction direction)	List<Move>	Melakukan scanning kemungkinan move secara vertikal.

3.7. File GameState.java

Method	Tipe	Deskripsi
isSolved(DataStructure dataStructure)	boolean	Mengecek apakah puzzle telah solved atau belum.

3.8. File Reader.java

Method	Tipe	Deskripsi
readFile(String filePath)	DataStructure	Membaca file dan menyimpan dalam DataStructure

readPieces(char[][] grid)	List<Piece>	Membaca piece yang ada di board
determineOrientation(List<Point> coordinates)	int	Menentukan orientasi horizontal atau vertikal

3.9. File Saver.java

Method	Tipe	Deskripsi
saveFile(String filePath , Solution solution)	void	Melakukan write solution pada file yang telah ditentukan pathnya.
writeBoard(BufferedWriter writer , DataStructure dataStructure)	void	Menulis Board Puzzle Ke Writer
writeLastBoard(BufferedWriter writer , DataStructure dataStructure)	void	Menulis Last Board Puzzle Ke Writer

3.9. File DataStructure.java

Atribut dan Type	Deskripsi
private int width;	Lebar Board
private int height;	Tinggi Board
private int pieceCount;	Banyak piece
private Point exit;	koordinat titik keluar
private Board board;	Board permainan yang berisi kumpulan Piece
private List<Piece> pieces;	Kumpulan Piece

3.10. File Board.java

Atribut dan Type	Deskripsi
private char[][] grid;	Matriks karakter
private int width;	Lebar Board
private int height;	Tinggi Board

3.11. File Piece.java

Atribut dan Type	Deskripsi
private char type;	Karakter/Huruf piece nya

private List<Point> coordinates;	Kumpulan koordinat tempat piece terletak
private int orientation;	Pergerakan horizontal atau vertikal

3.12. File Main.java

File ini akan memanggil fungsi-fungsi pada file lain dan akan menangani interaksi pengguna, mulai dari meminta input file, membaca konfigurasi file, menjalankan algoritma yang dipilih, hingga menampilkan hasil, dan memberikan opsi untuk menyimpan.

BAB 4
SOURCE CODE PROGRAM

4.1. Repository Github

Source Code lengkap dapat diakses melalui link github berikut:
https://github.com/RNXFreeze/Tucil3_13523021_13523095

4.2. Source Code

1. UCS.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095                        */
/* Nama - 1    : Muhammad Raihan Nazhim Oktana                            */
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB     */
/* Nama - 2    : Rafif Farras                                             */
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB     */
/* Tanggal     : Rabu, 21 Mei 2025                                        */
/* Tugas       : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)           */
/* File Path   : Tucil3_13523021_13523095/src/algorithm/UCS.java          */
/* Deskripsi   : F02 - UCS (Uniform Cost Search) Pathfinding Algorithm    */
/* PIC F02     : K01 - 13523021 - Muhammad Raihan Nazhim Oktana           */

// Package & Import
package algorithm;
import game.*;
import java.util.*;
import utils.*;

// Class Definition & Implementation
public class UCS {
    // DESKRIPSI
    // Public Class UCS

    // KAMUS
    // UCS : Constructor Class UCS
    // solveUCS : Function

    // PRIVATE ATTRIBUTES
    // None

    private UCS() {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class UCS

        // KAMUS LOKAL
        // None

        // ALGORITMA LOKAL
        // None
    }

    public static Solution solveUCS(String filePath , DataStructure
dataStructure , int num) {
        // DESKRIPSI LOKAL
        // Fungsi Utama UCS : Menyelesaikan pencarian jalur terpendek dari
dataStructure dan tipe heuristiknya.
        // UCS Blind Search : Tidak Menggunakan Heuristik Apapun (BFS
Uniform).

        // KAMUS LOKAL
        // dataStructure , nxt : Class DataStructure
        // pq : Priority Queue of Class Solution Sub Class Node
        // node , cur : Class Solution Sub Class Node
        // visited : HashSet of String
        // startTime , endTime : Long
        // num , cnt : Integer
        // time : Double
```

```
        // filePath , key : String

        // ALGORITMA LOKAL
        long startTime = System.nanoTime();
        PriorityQueue<Solution.Node> pq = new
PriorityQueue<>(Comparator.comparingInt(node -> node.gValue));
        Set<String> visited = new HashSet<>();
        pq.add(new Solution.Node(dataStructure , null , 0 , 0 , null));
        visited.add(GameLogic.boardKey(dataStructure));
        int cnt = 0;
        while (!pq.isEmpty()) {
            cnt++;
            Solution.Node cur = pq.poll();
            if (GameState.isSolved(cur.state)) {
                long endTime = System.nanoTime();
                double time = (endTime - startTime) / 1000000;
                return Solution.buildSolution(filePath , "Uniform Cost Search
(UCS)" , num , cnt , time , cur);
            } else {
                for (GameLogic.Move move :
GameLogic.generateMoves(cur.state)) {
                    DataStructure nxt = GameLogic.applyMove(cur.state ,
move);
                    String key = GameLogic.boardKey(nxt);
                    if (visited.add(key)) {
                        pq.add(new Solution.Node(nxt , move , cur.gValue + 1
, 0 , cur));
                    }
                }
            }
        }
        long endTime = System.nanoTime();
        double time = (endTime - startTime) / 1000000;
        return Solution.buildSolution(filePath , "Uniform Cost Search (UCS)"
, num , cnt , time , new Solution.Node(dataStructure , null , 0 , 0 ,
null));
    }
}
```

2. GBFS.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095
*/
/* Nama - 1    : Muhammad Raihan Nazhim Oktana
*/
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB
*/
/* Nama - 2    : Rafif Farras
*/
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB
*/
/* Tanggal     : Rabu, 21 Mei 2025
*/
/* Tugas       : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)
*/
/* File Path   : Tucil3_13523021_13523095/src/algorithm/GBFS.java
*/
/* Deskripsi   : F03 - GBFS (Greedy Best First Search) Pathfinding Algorithm
*/
/* PIC F03     : K01 - 13523021 - Muhammad Raihan Nazhim Oktana
*/

// Package & Import
package algorithm;
import game.*;
import java.util.*;
import utils.*;

// Class Definition & Implementation
```

```java
public class GBFS {
   // DESKRIPSI
   // Public Class GBFS

   // KAMUS
   // GBFS : Constructor Class GBFS
   // solveGBFS : Function

   // PRIVATE ATTRIBUTES
   // None

   private GBFS() {
       // DESKRIPSI LOKAL
       // Instansiasi Constructor Class GBFS

       // KAMUS LOKAL
       // None

       // ALGORITMA LOKAL
       // None
   }

   public static Solution solveGBFS(String filePath , DataStructure
dataStructure , int num) {
       // DESKRIPSI LOKAL
       // Fungsi Utama GBFS : Menyelesaikan pencarian jalur terpendek dari
dataStructure dan tipe heuristiknya.
       // GBFS : Greedy Best First Search berdasarkan heuristik yang
dihitung.

       // KAMUS LOKAL
       // dataStructure , nxt : Class DataStructure
       // pq : Priority Queue of Class Solution Sub Class Node
       // node , cur : Class Solution Sub Class Node
       // solveHeuristic : Function Class Heuristic
       // visited : HashSet of String
       // startTime , endTime : Long
       // num , cnt : Integer
       // time : Double
       // filePath , key : String

       // ALGORITMA LOKAL
       long startTime = System.nanoTime();
       int cnt = 0;
       Set<String> visited = new HashSet<>();
       PriorityQueue<Solution.Node> pq = new
PriorityQueue<>(Comparator.comparingInt(node -> node.hValue));
       pq.add(new Solution.Node(dataStructure , null , 0 ,
Heuristic.solveHeuristic(dataStructure , num) , null));
       while (!pq.isEmpty()) {
           Solution.Node cur = pq.poll();
           if (visited.add(GameLogic.boardKey(cur.state))) {
               cnt++;
               if (GameState.isSolved(cur.state)) {
                   long endTime = System.nanoTime();
                   double time = (endTime - startTime) / 1000000;
                   return Solution.buildSolution(filePath , "Greedy Best
First Search (GBFS)" , num , cnt , time , cur);
               } else {
                   for (GameLogic.Move move :
GameLogic.generateMoves(cur.state)) {
                       DataStructure nxt = GameLogic.applyMove(cur.state ,
move);
                       pq.add(new Solution.Node(nxt , move , 0 ,
Heuristic.solveHeuristic(nxt , num) , cur));
                   }
               }
           }
```

```
        }
        long endTime = System.nanoTime();
        double time = (endTime - startTime) / 1000000;
        return Solution.buildSolution(filePath , "Greedy Best First Search
(GBFS)" , num , cnt , time , new Solution.Node(dataStructure , null , 0 , 0
, null));
    }
}
```

3. AStar.java

```
/* Kelompok   : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1   : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1    : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB */
/* Nama - 2   : Rafif Farras                                         */
/* NIM - 2    : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB */
/* Tanggal    : Rabu, 21 Mei 2025                                    */
/* Tugas      : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)       */
/* File Path  : Tucil3_13523021_13523095/src/algorithm/AStar.java    */
/* Deskripsi  : F04 - AStar Pathfinding Algorithm                    */
/* PIC F04    : K02 - 13523095 - Rafif Farras                        */

// Package & Import
package algorithm;
import game.*;
import java.util.*;
import utils.*;

// Class Definition & Implementation
public class AStar {
    // DESKRIPSI
    // Public Class AStar

    // KAMUS
    // AStar : Constructor Class AStar
    // solveAStar : Function

    // PRIVATE ATTRIBUTES
    // None

    private AStar() {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class AStar

        // KAMUS LOKAL
        // None

        // ALGORITMA LOKAL
        // None
    }

    public static Solution solveAStar(String filePath , DataStructure
dataStructure , int num) {
        // DESKRIPSI LOKAL
        // Fungsi Utama AStar : Menyelesaikan pencarian jalur terpendek dari
dataStructure dan tipe heuristiknya.
        // AStar : A* Algorithm Search berdasarkan heuristik yang dihitung.

        // KAMUS LOKAL
        // dataStructure , nxt : Class DataStructure
        // pq : Priority Queue of Class Solution Sub Class Node
        // node1 , node2 , cur , childNode : Class Solution Sub Class Node
        // solveHeuristic : Function Class Heuristic
        // visited : HashSet of String
        // bestCost : HashMap of String To Integer
        // startTime , endTime : Long
        // num , cnt , f1 , f2 , gValue , hValue : Integer
        // time : Double
```

```
        // filePath , key : String

        // ALGORITMA LOKAL
        long startTime = System.nanoTime();
        int cnt = 0;
        Solution.Node res = null;
        Set<String> visited = new HashSet<>();
        Map<String , Integer> bestCost = new HashMap<>();
        PriorityQueue<Solution.Node> pq = new PriorityQueue<>((node1 , node2)
-> {
            int f1 = node1.solveF();
            int f2 = node2.solveF();
            if (f1 != f2) {
                return Integer.compare(f1 , f2);
            } else {
                return Integer.compare(node1.hValue , node2.hValue);
            }
        });
        String key = GameLogic.boardKey(dataStructure);
        pq.add(new Solution.Node(dataStructure , null , 0 ,
Heuristic.solveHeuristic(dataStructure , num) , null));
        bestCost.put(key , 0);
        while (!pq.isEmpty()) {
            Solution.Node cur = pq.poll();
            key = GameLogic.boardKey(cur.state);
            if (!visited.contains(key) || bestCost.get(key) > cur.gValue) {
                cnt++;
                visited.add(key);
                if (GameState.isSolved(cur.state)) {
                    if (res == null || cur.gValue < res.gValue) {
                        res = cur;
                    }
                } else {
                    for (GameLogic.Move move :
GameLogic.generateMoves(cur.state)) {
                        DataStructure nxt = GameLogic.applyMove(cur.state ,
move);
                        String nxtKey = GameLogic.boardKey(nxt);
                        int gValue = cur.gValue + 1;
                        if (!visited.contains(nxtKey) || gValue <
bestCost.getOrDefault(nxtKey , Integer.MAX_VALUE)) {
                            int hValue = Heuristic.solveHeuristic(nxt , num);
                            bestCost.put(nxtKey , gValue);
                            Solution.Node childNode = new Solution.Node(nxt ,
move , gValue , hValue , cur);
                            pq.add(childNode);
                        }
                    }
                }
            }
        }
        long endTime = System.nanoTime();
        double time = (endTime - startTime) / 1000000;
        if (res != null) {
            return Solution.buildSolution(filePath , "A-Star Search (A*)" ,
num , cnt , time , res);
        } else {
            return Solution.buildSolution(filePath , "A-Star Search (A*)" ,
num , cnt , time , new Solution.Node(dataStructure , null , 0 , 0 , null));
        }
    }
}
```

4. Heuristic.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1    : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB  */
```

```java
/* Nama - 2    : Rafif Farras                                          */
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB  */
/* Tanggal     : Rabu, 21 Mei 2025                                     */
/* Tugas       : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)        */
/* File Path   : Tucil3_13523021_13523095/src/algorithm/Heuristic.java */
/* Deskripsi   : F05 - Heuristic Pathfinding Algorithm                 */
/* PIC F05     : K01 - 13523021 - Muhammad Raihan Nazhim Oktana        */

// Package & Import
package algorithm;
import utils.*;

// Class Definition & Implementation
public class Heuristic {
    // DESKRIPSI
    // Public Class Heuristic

    // KAMUS
    // Heuristic : Constructor Class Heuristic
    // solveHeuristic , heuristic1 , heuristic2 , heuristic3 , blockingCount
, findPrimary , isBlocking : Function

    // PRIVATE ATTRIBUTES
    // None

    private Heuristic() {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class Heuristic

        // KAMUS LOKAL
        // None

        // ALGORITMA LOKAL
        // None
    }

    public static int solveHeuristic(DataStructure dataStructure , int num) {
        // DESKRIPSI LOKAL
        // Fungsi Utama Heuristik : Menghitung nilai heuristik dari
dataStructure dan tipe heuristiknya.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // heuristic1 , heuristic2 , heuristic3 : Function
        // num : Integer

        // ALGORITMA LOKAL
        return switch (num) {
            case 1 -> heuristic1(dataStructure);
            case 2 -> heuristic2(dataStructure);
            case 3 -> heuristic3(dataStructure);
            default -> 0;
        };
    }

    private static int heuristic1(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Heuristik 1 : Menghitung jumlah piece yang menghalangi primary
piece.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // blockingCount : Function

        // ALGORITMA LOKAL
        return blockingCount(dataStructure) + 1;
    }
```

```
    private static int heuristic2(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Heuristik 2 : Menghitung jumlah piece yang menghalangi primary
piece dan jarak yang tersisa.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // blockingCount , findPrimary : Function
        // piece : Class Piece
        // board : Class Board
        // blocker , rig , bot : Integer

        // ALGORITMA LOKAL
        int blocker = blockingCount(dataStructure);
        Piece piece = findPrimary(dataStructure);
        if (piece.getOrientation() == 0) {
            int rig =
piece.getCoordinates().stream().mapToInt(Point::getX).max().orElseThrow();
            return blocker + dataStructure.getExit().getX() - rig - 1;
        } else {
            int bot =
piece.getCoordinates().stream().mapToInt(Point::getY).max().orElseThrow();
            return blocker + dataStructure.getExit().getY() - bot - 1;
        }
    }

    private static int heuristic3(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Heuristik 3 : Menghitung jumlah piece yang menghalangi primary
piece dan ukuran setiap piece yang menghalangi tersebut.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // blockingCount , findPrimary , isBlocking : Function
        // piece , pc : Class Piece
        // blocker , extra : Integer

        // ALGORITMA LOKAL
        int blocker = blockingCount(dataStructure);
        Piece piece = findPrimary(dataStructure);
        int extra = dataStructure.getPieces().stream().filter(pc -> pc !=
piece && isBlocking(pc , piece)).mapToInt(pc ->
pc.getCoordinates().size()).sum();
        return blocker + extra;
    }

    private static int blockingCount(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Fungsi Bantuan : Menghitung jumlah piece yang menghalangi primary
piece.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // findPrimary : Function
        // piece : Class Piece
        // board : Class Board
        // cnt , row , col , rig , bot , x , y : Integer

        // ALGORITMA LOKAL
        Piece piece  = findPrimary(dataStructure);
        Board board = dataStructure.getBoard();
        int cnt = 0;
        if (piece.getOrientation() == 0) {
            int row = piece.getCoordinates().get(0).getY();
            int rig =
piece.getCoordinates().stream().mapToInt(Point::getX).max().orElseThrow();
            for (int x = rig + 1 ; x < board.getHeight() ; x++) {
                if (board.getCell(row , x) != '.') {
```

```
                    cnt++;
                }
            }
        } else {
            int col = piece.getCoordinates().get(0).getX();
            int bot =
piece.getCoordinates().stream().mapToInt(Point::getY).max().orElseThrow();
            for (int y = bot + 1 ; y < board.getHeight() ; y++) {
                if (board.getCell(y , col) != '.') {
                    cnt++;
                }
            }
        }
        return cnt;
    }

    private static Piece findPrimary(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Fungsi Bantuan : Mencari primary piece dari dataStructure.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // piece : Piece

        // ALGORITMA LOKAL
        return dataStructure.getPieces().stream().filter(piece ->
piece.getType() == 'P').findFirst().orElseThrow();
    }

    private static boolean isBlocking(Piece otherPiece , Piece primaryPiece)
{
        // DESKRIPSI LOKAL
        // Fungsi Bantuan : Mengecek apakah piece lain menghalangi primary
piece.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // primaryPiece , otherPiece , piece : Piece
        // exit : Class Point
        // prow , orow , pcol , ocol : Integer

        // ALGORITMA LOKAL
        if (primaryPiece.getOrientation() == 0) {
            int prow = primaryPiece.getCoordinates().get(0).getY();
            int orow = otherPiece.getCoordinates().get(0).getY();
            return ((prow == orow) &&
(otherPiece.getCoordinates().stream().anyMatch(piece -> piece.getX() >
primaryPiece.getCoordinates().stream().mapToInt(Point::getX).max().orElseThr
ow())));
        } else {
            int pcol = primaryPiece.getCoordinates().get(0).getX();
            int ocol = otherPiece.getCoordinates().get(0).getX();
            return pcol == ocol &&
otherPiece.getCoordinates().stream().anyMatch(piece -> piece.getY() >
primaryPiece.getCoordinates().stream().mapToInt(Point::getY).max().orElseThr
ow());
        }
    }
}
```

5. Solution.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095                      */
/* Nama - 1    : Muhammad Raihan Nazhim Oktana                          */
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB   */
/* Nama - 2    : Rafif Farras                                           */
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB   */
/* Tanggal     : Rabu, 21 Mei 2025                                      */
```

```java
/* Tugas      : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)        */
/* File Path  : Tucil3_13523021_13523095/src/algorithm/Solution.java  */
/* Deskripsi  : F06 - Solution Pathfinding Algorithm                  */
/* PIC F06    : K01 - 13523021 - Muhammad Raihan Nazhim Oktana        */

// Package & Import
package algorithm;
import game.*;
import java.util.*;
import utils.*;

// Class Definition & Implementation
public final class Solution {
   // DESKRIPSI
   // Public Class Solution

   // KAMUS
   // Solution : Constructor Class Solution
   // Node : Sub Class Node
   // getFilePath , getAlgorithm , getHeuristicId , getNodesVisited ,
getStepCount , getTime , getPath , getMoves : Procedure
   // setFilePath , setAlgorithm , setHeuristicId , setNodesVisited ,
setStepCount , setTime , setPath , setMoves : Procedure
   // buildSolution , displaySolution , displayColoredBoard ,
displayBoardWithHighlight : Procedure

   // PRIVATE ATTRIBUTES - Main
   private String filePath;
   private String algorithm;
   private int heuristicId;
   private int nodesVisited;
   private int stepCount;
   private double time;
   private List<DataStructure> path;
   private List<GameLogic.Move> moves;

   // PRIVATE ATTRIBUTES - ANSI Color Codes
   public static final String ANSI_RESET = "\u001B[0m";
   public static final String ANSI_RED = "\u001B[31m";
   public static final String ANSI_GREEN = "\u001B[32m";
   public static final String ANSI_YELLOW = "\u001B[33m";
   public static final String ANSI_PURPLE = "\u001B[35m";
   public static final String ANSI_CYAN = "\u001B[36m";
   public static final String ANSI_BOLD = "\u001B[1m";
   public static final String ANSI_BACKGROUND_RED = "\u001B[41m";
   public static final String ANSI_BACKGROUND_GREEN = "\u001B[42m";

   public Solution(String filePath , String algorithm , int heuristicId ,
int nodesVisited , double time , List<DataStructure> path ,
List<GameLogic.Move> moves) {
       // DESKRIPSI LOKAL
       // Instansiasi Constructor Class Solution

       // KAMUS LOKAL
       // filePath : String
       // algorithm : String
       // heuristicId : Integer
       // nodesVisited : Integer
       // time : Double
       // path : List of Class DataStructure
       // moves : List of Class GameLogic Sub Class Move

       // ALGORITMA LOKAL
       this.filePath = filePath;
       this.algorithm = algorithm;
       this.heuristicId = heuristicId;
       this.nodesVisited = nodesVisited;
       this.time = time;
```

```
        this.path = path;
        this.moves = moves;
        this.stepCount = moves.size();
    }

    public String getFilePath() {
        // DESKRIPSI LOKAL
        // Getter File Path

        // KAMUS LOKAL
        // filePath : String

        // ALGORITMA LOKAL
        return this.filePath;
    }

    public String getAlgorithm() {
        // DESKRIPSI LOKAL
        // Getter Algorithm

        // KAMUS LOKAL
        // algorithm : String

        // ALGORITMA LOKAL
        return this.algorithm;
    }

    public int getHeuristicId() {
        // DESKRIPSI LOKAL
        // Getter Heuristic ID

        // KAMUS LOKAL
        // heuristicId : Integer

        // ALGORITMA LOKAL
        return this.heuristicId;
    }

    public int getNodesVisited() {
        // DESKRIPSI LOKAL
        // Getter Nodes Visited

        // KAMUS LOKAL
        // nodesVisited : Integer

        // ALGORITMA LOKAL
        return this.nodesVisited;
    }

    public int getStepCount() {
        // DESKRIPSI LOKAL
        // Getter Depth

        // KAMUS LOKAL
        // stepCount : Integer

        // ALGORITMA LOKAL
        return this.stepCount;
    }

    public double getTime() {
        // DESKRIPSI LOKAL
        // Getter Time Millis

        // KAMUS LOKAL
        // time : Double

        // ALGORITMA LOKAL
```

```
        return this.time;
    }

    public List<DataStructure> getPath() {
        // DESKRIPSI LOKAL
        // Getter Path

        // KAMUS LOKAL
        // path : List of Class DataStructure

        // ALGORITMA LOKAL
        return this.path;
    }

    public List<GameLogic.Move> getMoves() {
        // DESKRIPSI LOKAL
        // Getter Moves

        // KAMUS LOKAL
        // moves : List of Class GameLogic Sub Class Move

        // ALGORITMA LOKAL
        return this.moves;
    }

    public void setFilePath(String filePath) {
        // DESKRIPSI LOKAL
        // Setter File Path

        // KAMUS LOKAL
        // filePath : String

        // ALGORITMA LOKAL
        this.filePath = filePath;
    }

    public void setAlgorithm(String algorithm) {
        // DESKRIPSI LOKAL
        // Setter Algorithm

        // KAMUS LOKAL
        // algorithm : String

        // ALGORITMA LOKAL
        this.algorithm = algorithm;
    }

    public void setHeuristicId(int heuristicId) {
        // DESKRIPSI LOKAL
        // Setter Heuristic ID

        // KAMUS LOKAL
        // heuristicId : Integer

        // ALGORITMA LOKAL
        this.heuristicId = heuristicId;
    }

    public void setNodesVisited(int nodesVisited) {
        // DESKRIPSI LOKAL
        // Setter Nodes Visited

        // KAMUS LOKAL
        // nodesVisited : Integer

        // ALGORITMA LOKAL
        this.nodesVisited = nodesVisited;
    }
```

```
    public void setStepCount(int stepCount) {
        // DESKRIPSI LOKAL
        // Setter Step Count

        // KAMUS LOKAL
        // stepCount : Integer

        // ALGORITMA LOKAL
        this.stepCount = stepCount;
    }

    public void setTime(double time) {
        // DESKRIPSI LOKAL
        // Setter Time Millis

        // KAMUS LOKAL
        // time : Double

        // ALGORITMA LOKAL
        this.time = time;
    }

    public void setPath(List<DataStructure> path) {
        // DESKRIPSI LOKAL
        // Setter Path

        // KAMUS LOKAL
        // path : List of Class DataStructure

        // ALGORITMA LOKAL
        this.path = path;
    }

    public void setMoves(List<GameLogic.Move> moves) {
        // DESKRIPSI LOKAL
        // Setter Moves

        // KAMUS LOKAL
        // moves : List of Class GameLogic Sub Class Move

        // ALGORITMA LOKAL
        this.moves = moves;
    }

    public static class Node {
        // DESKRIPSI SUBCLASS
        // Public Class Node

        // KAMUS SUBCLASS
        // Node : Constructor Class Solution Sub Class Node
        // solveF : Function
        // getState , getMove , getGValue , getHValue , getParent : Procedure
        // setState , setMove , setGValue , setHValue , setParent : Procedure

        // PRIVATE ATTRIBUTES SUBCLASS
        public DataStructure state;
        public GameLogic.Move move;
        public int gValue;
        public int hValue;
        public Node parent;

        public Node(DataStructure state , GameLogic.Move move , int gValue ,
int hValue , Node parent) {
            // DESKRIPSI SUBCLASS LOKAL
            // Instansiasi Constructor Class Node

            // KAMUS SUBCLASS LOKAL
```

```
        // state : Class DataStructure
        // move : Class GameLogic Sub Class Move
        // parent : Class Solution Sub Class Node
        // gValue , hValue : Integer

        // ALGORITMA SUBCLASS LOKAL
        this.state = state;
        this.move = move;
        this.gValue = gValue;
        this.hValue = hValue;
        this.parent = parent;
    }

    public DataStructure getState() {
        // DESKRIPSI LOKAL
        // Getter State

        // KAMUS LOKAL
        // state : Class DataStructure

        // ALGORITMA LOKAL
        return this.state;
    }

    public GameLogic.Move getMove() {
        // DESKRIPSI LOKAL
        // Getter Move

        // KAMUS LOKAL
        // move : Class GameLogic Sub Class Move

        // ALGORITMA LOKAL
        return this.move;
    }

    public int getGValue() {
        // DESKRIPSI LOKAL
        // Getter G Value

        // KAMUS LOKAL
        // gValue : Integer

        // ALGORITMA LOKAL
        return this.gValue;
    }

    public int getHValue() {
        // DESKRIPSI LOKAL
        // Getter H Value

        // KAMUS LOKAL
        // hValue : Integer

        // ALGORITMA LOKAL
        return this.hValue;
    }

    public Node getParent() {
        // DESKRIPSI LOKAL
        // Getter Parent

        // KAMUS LOKAL
        // parent : Class Solution Sub Class Node

        // ALGORITMA LOKAL
        return this.parent;
    }
```

```
    public void setState(DataStructure state) {
        // DESKRIPSI LOKAL
        // Setter State

        // KAMUS LOKAL
        // state : Class DataStructure

        // ALGORITMA LOKAL
        this.state = state;
    }

    public void setMove(GameLogic.Move move) {
        // DESKRIPSI LOKAL
        // Setter Move

        // KAMUS LOKAL
        // move : Class GameLogic Sub Class Move

        // ALGORITMA LOKAL
        this.move = move;
    }

    public void setGValue(int gValue) {
        // DESKRIPSI LOKAL
        // Setter G Value

        // KAMUS LOKAL
        // gValue : Integer

        // ALGORITMA LOKAL
        this.gValue = gValue;
    }

    public void setHValue(int hValue) {
        // DESKRIPSI LOKAL
        // Setter H Value

        // KAMUS LOKAL
        // hValue : Integer

        // ALGORITMA LOKAL
        this.hValue = hValue;
    }

    public void setParent(Node parent) {
        // DESKRIPSI LOKAL
        // Setter Parent

        // KAMUS LOKAL
        // parent : Class Solution Sub Class Node

        // ALGORITMA LOKAL
        this.parent = parent;
    }

    public int solveF() {
        // DESKRIPSI SUBCLASS LOKAL
        // Menghitung nilai F pada algoritma pencarian A-Star

        // KAMUS SUBCLASS LOKAL
        // gValue , hValue : Integer

        // ALGORITMA SUBCLASS LOKAL
        return this.gValue + this.hValue;
    }
}
```

```
   public static Solution buildSolution(String filePath , String algorithm ,
int heuristicId , int visitedCount , double time , Node goal) {
       // DESKRIPSI LOKAL
       // Membangun tipe curDataStructure Solution dari hasil algoritma
pencarian

       // KAMUS LOKAL
       // moves : List of Class GameLogic Sub Class Move
       // boards : List of Class DataStructure
       // node , goal : Sub Class Node
       // heuristicId , visitedCount : Integer
       // algorithm : String
       // time : Double

       // ALGORITMA LOKAL
       LinkedList<DataStructure> boards = new LinkedList<>();
       LinkedList<GameLogic.Move> moves = new LinkedList<>();
       for (Node node = goal ; node != null ; node = node.parent) {
           boards.addFirst(node.state);
           if (node.move != null) {
               moves.addFirst(node.move);
           }
       }
       return new Solution(filePath , algorithm , heuristicId , visitedCount
, time , boards , moves);
   }

   public void displaySolution() {
       // DESKRIPSI LOKAL
       // Display Solution To CLI Terminal

       // KAMUS LOKAL
       // algorithm : String
       // heuristicId , nodesVisited , i : Integer
       // time : Double
       // path : List of Class DataStructure
       // moves : List of Class GameLogic Sub Class Move

       // ALGORITMA LOKAL
       System.out.println(ANSI_BOLD +
"==================================================" + ANSI_RESET);
       System.out.println(ANSI_BOLD + "INFORMATION SOLUTION RESULT :" +
ANSI_RESET);
       System.out.printf("File Path    : %s\n" , this.filePath);
       System.out.printf("Algorithm    : %s%s%s\n" , ANSI_CYAN ,
this.algorithm , ANSI_RESET);
       if (this.heuristicId == 0) {
           System.out.println("Heuristic    : None");
       } else {
           System.out.printf("Heuristic    : %s%d%s\n" , ANSI_CYAN ,
this.heuristicId , ANSI_RESET);
       }
       System.out.printf("Step Count   : %s%d Step%s\n" , ANSI_YELLOW ,
this.stepCount , ANSI_RESET);
       System.out.printf("Visited Node : %s%d Node%s\n" , ANSI_YELLOW ,
this.nodesVisited , ANSI_RESET);
       System.out.printf("Time Usage   : %s%d ms%s\n" , ANSI_YELLOW , (int)
this.time , ANSI_RESET);
       if (this.moves.isEmpty()) {
           System.out.println("Success      : " + ANSI_RED + "NO" +
ANSI_RESET);
       } else {
           System.out.println("Success      : " + ANSI_GREEN + "YES" +
ANSI_RESET);
       }
       System.out.println(ANSI_BOLD +
"==================================================" + ANSI_RESET);
       if (this.moves.isEmpty()) {
```

```
            System.out.println("Display Board Awal & Akhir :");
            this.path.get(0).displayBoard();
        } else {
            System.out.println("Display Board Awal :");
            this.path.get(0).displayBoard();
            for (int i = 0 ; i < this.moves.size() - 1 ; i++) {
                System.out.printf("\n%sMOVE %d : %s%s\n" , ANSI_BOLD +
ANSI_PURPLE , i + 1 , this.moves.get(i) , ANSI_RESET);
                displayBoardWithHighlight(this.path.get(i) , this.path.get(i
+ 1) , this.moves.get(i));
            }
            System.out.printf("\n%sMOVE %d : P - OUT %s (%d STEP)%s\n" ,
ANSI_BOLD + ANSI_PURPLE , this.moves.size() , switch
(this.moves.get(this.moves.size() - 1).getDirection()) {
                case UP -> "UP";
                case DOWN -> "DOWN";
                case RIGHT -> "RIGHT";
                case LEFT -> "LEFT";
                default -> "UNKNOWN";
            } , this.path.get(0).getPieces().stream().filter(pc ->
pc.getType() == 'P').findFirst().orElseThrow().solveSize() +
this.moves.get(this.moves.size() - 1).getStepCount() , ANSI_RESET);
            this.path.get(this.moves.size() - 1).displayLastBoard();
        }
        System.out.println(ANSI_BOLD +
"================================================" + ANSI_RESET);
        System.out.println(ANSI_BOLD + "RECALL INFORMATION SOLUTION RESULT :"
+ ANSI_RESET);
        System.out.printf("File Path    : %s\n" , this.filePath);
        System.out.printf("Algorithm    : %s%s%s\n" , ANSI_CYAN ,
this.algorithm , ANSI_RESET);
        if (this.heuristicId == 0) {
            System.out.println("Heuristic    : None");
        } else {
            System.out.printf("Heuristic    : %s%d%s\n" , ANSI_CYAN ,
this.heuristicId , ANSI_RESET);
        }
        System.out.printf("Step Count   : %s%d Step%s\n" , ANSI_YELLOW ,
this.stepCount , ANSI_RESET);
        System.out.printf("Visited Node : %s%d Node%s\n" , ANSI_YELLOW ,
this.nodesVisited , ANSI_RESET);
        System.out.printf("Time Usage   : %s%d ms%s\n" , ANSI_YELLOW , (int)
this.time , ANSI_RESET);
        if (this.moves.isEmpty()) {
            System.out.println("Success      : " + ANSI_RED + "NO" +
ANSI_RESET);
        } else {
            System.out.println("Success      : " + ANSI_GREEN + "YES" +
ANSI_RESET);
        }
    }

    private void displayColoredBoard(DataStructure curDataStructure ,
DataStructure prevDataStructure , GameLogic.Move move) {
        // DESKRIPSI LOKAL
        // Display Board From Data Structure To CLI Terminal With Color

        // KAMUS LOKAL
        // curDataStructure , prevDataStructure : Class DataStructure
        // move : Class GameLogic Sub Class Move
        // board : Class Board
        // exit : Class Point
        // piece : Class Piece
        // movingPiece , cell : Character
        // width , height , i , j : Integer
        // oldPositions , newPositions : Set of Class Point

        // ALGORITMA LOKAL
```

```
        int width = curDataStructure.getWidth();
        int height = curDataStructure.getHeight();
        Point exit = curDataStructure.getExit();
        Board board = curDataStructure.getBoard();
        Set<Point> oldPositions = new HashSet<>();
        Set<Point> newPositions = new HashSet<>();
        if (prevDataStructure != null && move != null) {
            char movingPiece = move.getIdType();
            for (Piece piece : prevDataStructure.getPieces()) {
                if (piece.getType() == movingPiece) {
                    oldPositions.addAll(piece.getCoordinates());
                    break;
                }
            }
            for (Piece piece : curDataStructure.getPieces()) {
                if (piece.getType() == movingPiece) {
                    newPositions.addAll(piece.getCoordinates());
                    break;
                }
            }
        }
        if (exit.getX() == -1) {
            for (int i = 0 ; i < height ; i++) {
                if (height - 1 - exit.getY() == i) {
                    System.out.print(ANSI_BOLD + ANSI_PURPLE + "K " +
ANSI_RESET);
                } else {
                    System.out.print("  ");
                }
                for (int j = 0 ; j < width ; j++) {
                    Point point = new Point(j , i);
                    char cell = board.getCell(j , i);
                    if (prevDataStructure != null &&
oldPositions.contains(point) && !newPositions.contains(point)) {
                        System.out.print(ANSI_BACKGROUND_RED + " " +
ANSI_RESET + " ");
                    } else if (prevDataStructure != null &&
newPositions.contains(point)) {
                        if (cell == 'P') {
                            System.out.print(ANSI_BACKGROUND_GREEN +
ANSI_BOLD + ANSI_CYAN + cell + " " + ANSI_RESET);
                        } else {
                            System.out.print(ANSI_BACKGROUND_GREEN + cell + "
" + ANSI_RESET);
                        }
                    } else if (cell == 'P') {
                        System.out.print(ANSI_BOLD + ANSI_CYAN + cell + " " +
ANSI_RESET);
                    } else {
                        System.out.print(cell + " ");
                    }
                }
                System.out.println();
            }
        } else if (exit.getY() == -1) {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    Point point = new Point(j , i);
                    char cell = board.getCell(j , i);
                    if (prevDataStructure != null &&
oldPositions.contains(point) && !newPositions.contains(point)) {
                        System.out.print(ANSI_BACKGROUND_RED + " " +
ANSI_RESET + " ");
                    } else if (prevDataStructure != null &&
newPositions.contains(point)) {
                        if (cell == 'P') {
                            System.out.print(ANSI_BACKGROUND_GREEN +
ANSI_BOLD + ANSI_CYAN + cell + " " + ANSI_RESET);
```

```
                        } else {
                            System.out.print(ANSI_BACKGROUND_GREEN + cell + "
" + ANSI_RESET);
                        }
                    } else if (cell == 'P') {
                        System.out.print(ANSI_BOLD + ANSI_CYAN + cell + " " +
ANSI_RESET);
                    } else {
                        System.out.print(cell + " ");
                    }
                }
                System.out.println();
            }
            for (int i = 0 ; i < exit.getX() ; i++) {
                System.out.print("  ");
            }
            System.out.println(ANSI_BOLD + ANSI_PURPLE + "K" + ANSI_RESET);
        } else if (exit.getY() == height) {
            for (int i = 0 ; i < exit.getX() ; i++) {
                System.out.print("  ");
            }
            System.out.println(ANSI_BOLD + ANSI_PURPLE + "K" + ANSI_RESET);
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    Point point = new Point(j , i);
                    char cell = board.getCell(j , i);
                    if (prevDataStructure != null &&
oldPositions.contains(point) && !newPositions.contains(point)) {
                        System.out.print(ANSI_BACKGROUND_RED + " " +
ANSI_RESET + " ");
                    } else if (prevDataStructure != null &&
newPositions.contains(point)) {
                        if (cell == 'P') {
                            System.out.print(ANSI_BACKGROUND_GREEN +
ANSI_BOLD + ANSI_CYAN + cell + " " + ANSI_RESET);
                        } else {
                            System.out.print(ANSI_BACKGROUND_GREEN + cell + "
" + ANSI_RESET);
                        }
                    } else if (cell == 'P') {
                        System.out.print(ANSI_BOLD + ANSI_CYAN + cell + " " +
ANSI_RESET);
                    } else {
                        System.out.print(cell + " ");
                    }
                }
                System.out.println();
            }
        } else {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    Point point = new Point(j , i);
                    char cell = board.getCell(j , i);
                    if (prevDataStructure != null &&
oldPositions.contains(point) && !newPositions.contains(point)) {
                        System.out.print(ANSI_BACKGROUND_RED + " " +
ANSI_RESET + " ");
                    } else if (prevDataStructure != null &&
newPositions.contains(point)) {
                        if (cell == 'P') {
                            System.out.print(ANSI_BACKGROUND_GREEN +
ANSI_BOLD + ANSI_CYAN + cell + " " + ANSI_RESET);
                        } else {
                            System.out.print(ANSI_BACKGROUND_GREEN + cell + "
" + ANSI_RESET);
                        }
                    } else if (cell == 'P') {
```

```
                        System.out.print(ANSI_BOLD + ANSI_CYAN + cell + " " +
ANSI_RESET);
                    } else {
                        System.out.print(cell + " ");
                    }
                }
                if (height - 1 - exit.getY() == i) {
                    System.out.print(ANSI_BOLD + ANSI_PURPLE + "K" +
ANSI_RESET);
                }
                System.out.println();
            }
        }
    }

    private void displayBoardWithHighlight(DataStructure prevDataStructure ,
DataStructure curDataStructure , GameLogic.Move move) {
        // DESKRIPSI LOKAL
        // Display Board From Data Structure To CLI Terminal With Color And
Notes

        // KAMUS LOKAL
        // curDataStructure , prevDataStructure : Class DataStructure
        // move : Class GameLogic Sub Class Move

        // ALGORITMA LOKAL
        displayColoredBoard(curDataStructure , prevDataStructure , move);
        System.out.println("\nNotes :");
        System.out.println(ANSI_BACKGROUND_RED + " " + ANSI_RESET + "  : Old
Position Piece " + move.getIdType());
        System.out.println(ANSI_BACKGROUND_GREEN + move.getIdType() + " " +
ANSI_RESET + " : New Position Piece " + move.getIdType());
        System.out.println(ANSI_BOLD + ANSI_CYAN + "P" + ANSI_RESET + "  :
Primary Piece");
        System.out.println(ANSI_BOLD + ANSI_PURPLE + "K" + ANSI_RESET + "  :
Exit Position");
    }
}
```

6. GameLogic.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1    : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB  */
/* Nama - 2    : Rafif Farras                                         */
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB  */
/* Tanggal     : Rabu, 21 Mei 2025                                    */
/* Tugas       : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)       */
/* File Path   : Tucil3_13523021_13523095/src/game/GameLogic.java     */
/* Deskripsi   : F10 - Game Logic Program                             */
/* PIC F10     : K01 - 13523021 - Muhammad Raihan Nazhim Oktana       */

// Package & Import
package game;
import java.util.*;
import utils.*;

// Class Definition & Implementation
public final class GameLogic {
    // DESKRIPSI
    // Public Class GameLogic

    // KAMUS
```

```
    // GameLogic : Constructor Class GameLogic
    // Move : Subclass Movement
    // generateMoves , applyMove , scanHorizontal , scanVertical : Procedure

    // PRIVATE ATTRIBUTES
    // None

    public enum Direction {
        // DESKRIPSI LOKAL
        // Instansiasi Enumeration Direction

        // KAMUS LOKAL
        // LEFT , RIGHT , UP , DOWN : Enumeration

        // ALGORITMA LOKAL
        LEFT , RIGHT , UP , DOWN
    }

    public static class Move {
        // DESKRIPSI SUBCLASS
        // Public Class Move

        // KAMUS SUBCLASS
        // Move : Constructor Sub Class Move
        // getIdType , getDirection , getStepCount , setIdType , setDirection
, setStepCount , toString : Procedure

        // PRIVATE ATTRIBUTES SUBCLASS
        private char idType;
        private Direction direction;
        private int stepCount;

        public Move(char idType , Direction direction , int stepCount) {
            // DESKRIPSI SUBCLASS LOKAL
            // Instansiasi Constructor Class Move

            // KAMUS SUBCLASS LOKAL
            // idType : Character
            // direction : Enumeration Direction
            // stepCount : Integer

            // ALGORITMA SUBCLASS LOKAL
            this.idType = idType;
            this.direction = direction;
            this.stepCount = stepCount;
        }

        public char getIdType() {
            // DESKRIPSI SUBCLASS LOKAL
            // Getter ID Type

            // KAMUS SUBCLASS LOKAL
            // idType : Character

            // ALGORITMA SUBCLASS LOKAL
            return this.idType;
        }

        public Direction getDirection() {
            // DESKRIPSI SUBCLASS LOKAL
            // Getter Direction

            // KAMUS SUBCLASS LOKAL
            // direction : Enumeration Direction

            // ALGORITMA SUBCLASS LOKAL
            return this.direction;
        }
```

```
    public int getStepCount() {
        // DESKRIPSI SUBCLASS LOKAL
        // Getter Step Count

        // KAMUS SUBCLASS LOKAL
        // stepCount : Integer

        // ALGORITMA SUBCLASS LOKAL
        return this.stepCount;
    }

    public void setIdType(char idType) {
        // DESKRIPSI SUBCLASS LOKAL
        // Setter ID Type

        // KAMUS SUBCLASS LOKAL
        // idType : Character

        // ALGORITMA SUBCLASS LOKAL
        this.idType = idType;
    }

    public void setDirection(Direction direction) {
        // DESKRIPSI SUBCLASS LOKAL
        // Setter Direction

        // KAMUS SUBCLASS LOKAL
        // direction : Enumeration Direction

        // ALGORITMA SUBCLASS LOKAL
        this.direction = direction;
    }

    public void setStepCount(int stepCount) {
        // DESKRIPSI SUBCLASS LOKAL
        // Setter Step Count

        // KAMUS SUBCLASS LOKAL
        // stepCount : Integer

        // ALGORITMA SUBCLASS LOKAL
        this.stepCount = stepCount;
    }

    @Override
    public String toString() {
        // DESKRIPSI SUBCLASS LOKAL
        // Display Move To String

        // KAMUS SUBCLASS LOKAL
        // res : String
        // idType : Character
        // direction : Enumeration Direction
        // stepCount : Integer

        // ALGORITMA SUBCLASS LOKAL
        String res = this.idType + " - ";
        res += switch (this.direction) {
            case LEFT -> "LEFT ";
            case RIGHT -> "RIGHT ";
            case UP -> "UP ";
            case DOWN -> "DOWN ";
            default -> "UNKNOWN ";
        };
        res = res + "(" + this.stepCount + " STEP)";
        return res;
    }
```

```
    }

    public static String boardKey(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Membangun string linear dari dataStructure untuk digunakan sebagai
key pada HashSet & HashMap

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // sb : StringBuilder
        // row : Array of Character
        // grid : Matrix of Character

        // ALGORITMA LOKAL
        StringBuilder sb = new StringBuilder();
        char[][] grid = dataStructure.getBoard().getGrid();
        for (char[] row : grid) {
            sb.append(row);
        }
        return sb.toString();
    }


    private GameLogic() {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class GameLogic

        // KAMUS LOKAL
        // None

        // ALGORITMA LOKAL
        // None
    }

    public static List<Move> generateMoves(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Melakukan generate semua kemungkinan gerakan piece.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // listMoves : List of Sub Class Move
        // grid : Matrix of Character
        // piece : Class Piece

        // ALGORITMA LOKAL
        List<Move> listMoves = new ArrayList<>();
        char[][] grid = dataStructure.getBoard().getGrid();
        for (Piece piece : dataStructure.getPieces()) {
            if (piece.getType() != '.') {
                if (piece.getOrientation() == 0) {
                    listMoves.addAll(scanHorizontal(piece , grid ,
Direction.LEFT));
                    listMoves.addAll(scanHorizontal(piece , grid ,
Direction.RIGHT));
                } else {
                    listMoves.addAll(scanVertical(piece , grid ,
Direction.UP));
                    listMoves.addAll(scanVertical(piece , grid ,
Direction.DOWN));
                }
            }
        }
        return listMoves;
    }

    public static DataStructure applyMove(DataStructure dataStructure , Move
move) {
        // DESKRIPSI LOKAL
```

```
        // Melakukan suatu gerakan dan mengupdate board permainan.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // newBoard : Class Board
        // newCoordinates : List of Class Point
        // move : Sub Class Move
        // movedPiece , piece : Class Piece
        // newPieces : List of Class Piece
        // nx , ny : Integer

        // ALGORITMA LOKAL
        Board newBoard = new Board(dataStructure.getBoard());
        List<Piece> newPieces = new ArrayList<>();
        Piece movedPiece = null;
        for (Piece piece : dataStructure.getPieces()) {
            if (piece.getType() == move.idType) {
                movedPiece = piece;
            } else {
                newPieces.add(piece);
            }
        }
        if (movedPiece == null) {
            return dataStructure;
        } else {
            for (Point point : movedPiece.getCoordinates()) {
                newBoard.setCell(point.getX() , point.getY() , '.');
            }
            List<Point> newCoordinates = new ArrayList<>();
            for (Point point : movedPiece.getCoordinates()) {
                int nx = point.getX();
                int ny = point.getY();
                nx += switch (move.direction) {
                    case LEFT -> -move.stepCount;
                    case RIGHT -> move.stepCount;
                    case UP -> 0;
                    case DOWN -> 0;
                    default -> 0;
                };
                ny += switch (move.direction) {
                    case LEFT -> 0;
                    case RIGHT -> 0;
                    case UP -> -move.stepCount;
                    case DOWN -> move.stepCount;
                    default -> 0;
                };
                newCoordinates.add(new Point(nx , ny));
                newBoard.setCell(nx , ny , move.idType);
            }
            newPieces.add(new Piece(move.idType , newCoordinates ,
movedPiece.getOrientation()));
            return new DataStructure(dataStructure.getWidth() ,
dataStructure.getHeight() , dataStructure.getPieceCount() ,
dataStructure.getExit() , newBoard , newPieces);
        }
    }

    private static List<Move> scanHorizontal(Piece piece , char[][] grid ,
Direction direction) {
        // DESKRIPSI LOKAL
        // Melakukan scanning kemungkinan move secara horizontal.

        // KAMUS LOKAL
        // row , mnx , mxx , cnt , x : Integer
        // res : List of Sub Class Move
        // grid : Matrix of Character
        // direction : Sub Class Move
```

```
        // ALGORITMA LOKAL
        List<Move> res = new ArrayList<>();
        int row = piece.getCoordinates().get(0).getY();
        int mnx =
piece.getCoordinates().stream().mapToInt(Point::getX).min().orElseThrow();
        int mxx =
piece.getCoordinates().stream().mapToInt(Point::getX).max().orElseThrow();
        int cnt = 1;
        if (direction == Direction.LEFT) {
            for (int x = mnx - 1 ; x >= 0 ; x--) {
                if (grid[row][x] == '.') {
                    res.add(new Move(piece.getType() , direction , cnt));
                    cnt++;
                } else {
                    break;
                }
            }
        } else {
            for (int x = mxx + 1 ; x < grid[0].length ; x++) {
                if (grid[row][x] == '.') {
                    res.add(new Move(piece.getType() , direction , cnt));
                    cnt++;
                } else {
                    break;
                }
            }
        }
        return res;
    }

    private static List<Move> scanVertical(Piece piece , char[][] grid ,
Direction direction) {
        // DESKRIPSI LOKAL
        // Melakukan scanning kemungkinan move secara vertical.

        // KAMUS LOKAL
        // col , mny , mxy , cnt , y : Integer
        // res : List of Sub Class Move
        // grid : Matrix of Character
        // direction : Sub Class Move

        // ALGORITMA LOKAL
        List<Move> res = new ArrayList<>();
        int col = piece.getCoordinates().get(0).getX();
        int mny =
piece.getCoordinates().stream().mapToInt(Point::getY).min().orElseThrow();
        int mxy =
piece.getCoordinates().stream().mapToInt(Point::getY).max().orElseThrow();
        int cnt = 1;
        if (direction == Direction.DOWN) {
            for (int y = mxy + 1 ; y < grid.length ; y++) {
                if (grid[y][col] == '.') {
                    res.add(new Move(piece.getType() , direction , cnt));
                    cnt++;
                } else {
                    break;
                }
            }
        } else {
            for (int y = mny - 1 ; y >= 0 ; y--) {
                if (grid[y][col] == '.') {
                    res.add(new Move(piece.getType() , direction , cnt));
                    cnt++;
                } else {
                    break;
                }
            }
        }
```

```
        return res;
    }
}
```

7. GameState.java

```
/* Kelompok   : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1   : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1    : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB */
/* Nama - 2   : Rafif Farras                                         */
/* NIM - 2    : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB */
/* Tanggal    : Rabu, 21 Mei 2025                                    */
/* Tugas      : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)       */
/* File Path  : Tucil3_13523021_13523095/src/game/GameState.java     */
/* Deskripsi  : F09 - Game State Program                             */
/* PIC F09    : K01 - 13523021 - Muhammad Raihan Nazhim Oktana       */

// Package & Import
package game;
import utils.*;

// Class Definition & Implementation
public class GameState {
    // DESKRIPSI
    // Public Class GameState

    // KAMUS
    // GameState : Constructor Class GameState
    // isSolved : Function

    // PRIVATE ATTRIBUTES
    // None

    private GameState() {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class GameState

        // KAMUS LOKAL
        // None

        // ALGORITMA LOKAL
        // None
    }

    public static boolean isSolved(DataStructure dataStructure) {
        // DESKRIPSI LOKAL
        // Mengecek apakah puzzle telah solved atau belum.

        // KAMUS LOKAL
        // dataStructure : Class DataStructure
        // piece , pc : Class Piece
        // row , col , rig , lft , top , bot : Integer

        // ALGORITMA LOKAL
        Piece piece = dataStructure.getPieces().stream().filter(pc ->
pc.getType() == 'P').findFirst().orElseThrow();
        Point exit = dataStructure.getExit();
        if (piece.getOrientation() == 0) {
            int row = piece.getCoordinates().get(0).getY();
            int rig =
piece.getCoordinates().stream().mapToInt(Point::getX).max().orElseThrow();
            int lft =
piece.getCoordinates().stream().mapToInt(Point::getX).min().orElseThrow();
            if (exit.getX() == -1) {
                return ((exit.getY() == dataStructure.getHeight() - 1 - row)
&& (lft <= exit.getX() + 1));
            } else {
```

```
                return ((exit.getY() == dataStructure.getHeight() - 1 - row)
&& (rig >= exit.getX() - 1));
            }
        } else {
            int col = piece.getCoordinates().get(0).getX();
            int bot =
piece.getCoordinates().stream().mapToInt(Point::getY).max().orElseThrow();
            int top =
piece.getCoordinates().stream().mapToInt(Point::getY).min().orElseThrow();
            if (exit.getY() == -1) {
                return ((exit.getX() == col) && (dataStructure.getHeight() -
1 - bot <= exit.getY() + 1));
            } else {
                return ((exit.getX() == col) && (dataStructure.getHeight() -
1 - top >= exit.getY() - 1));
            }
        }
    }
}
```

8. Reader.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1    : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB  */
/* Nama - 2    : Rafif Farras                                         */
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB  */
/* Tanggal     : Rabu, 21 Mei 2025                                    */
/* Tugas       : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)        */
/* File Path   : Tucil3_13523021_13523095/src/utils/Reader.java       */
/* Deskripsi   : F08 - File Reader Utility                            */
/* PIC F08     : K02 - 13523095 - Rafif Farras                        */

// Package & Import
package utils;
import java.io.*;
import java.nio.file.*;
import java.util.*;

// Class Definition & Implementation
public class Reader {
    // DESKRIPSI
    // Public Class Reader

    // KAMUS
    // Reader : Constructor Class Reader
    // readFile , readPieces , determineOrientation : Procedure

    // PRIVATE ATTRIBUTES
    // None

    private Reader() {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class Reader

        // KAMUS LOKAL
        // None

        // ALGORITMA LOKAL
        // None
    }

    public static DataStructure readFile(String filePath) throws IOException
{
        // DESKRIPSI LOKAL
        // Melakukan read pada file yang telah ditentukan pathnya.

        // KAMUS LOKAL
        // filePath , line , firstLine , lines , trimmed , row : String
```

```
        // path : Path
        // reader : Java IO BufferedReader
        // dimensions : Array of String
        // rawLines , trimmedLines : List of String
        // pieces : List of Class Piece
        // grid : Matrix of Character
        // board : Class Board
        // exit : Class Point
        // c : Character
        // width , height , pieceCount , newY , x , i , j : Integer

        // ALGORITMA LOKAL
        Path path = Paths.get(filePath);
        try (BufferedReader reader = Files.newBufferedReader(path)) {
            /* ========================================== */
            /* ==========  DEBUGGED AREA  ============== */
            /* ========================================== */
            String[] dimensions = reader.readLine().trim().split("\\s+");
            if (dimensions.length != 2) {
                throw new IOException("Baris Ke-1 Harus Berisi 2 Bilangan
Bulat : Width & Height (A & B)");
            } else {
                int width , height;
                try {
                    width  = Integer.parseInt(dimensions[0]);
                    height = Integer.parseInt(dimensions[1]);
                } catch (NumberFormatException e) {
                    throw new IOException("Width & Height Harus Bilangan
Bulat Positif");
                }
                if (width <= 0 || height <= 0) {
                    throw new IOException("Width & Height Harus Bilangan
Bulat Positif");
                } else {
                    int pieceCount;
                    try {
                        pieceCount =
Integer.parseInt(reader.readLine().trim());
                    } catch (NumberFormatException e) {
                        throw new IOException("Baris Ke-2 Harus Berisi 1
Bilangan Bulat : Piece Count (N)");
                    }
                    int area = width * height;
                    int maxn = (area - 2) / 2;
                    if (pieceCount > maxn || pieceCount >= 25) {
                        throw new IOException("Nilai N Harus Kurang Dari 1/2
* (area - 2) & Kurang dari 25.");
                    }
                    /* ========================================== */
                    /* ==========  UNDEBUGGED AREA  ============== */
                    /* ========================================== */
                    List<String> rawLines = new ArrayList<>();
                    String line;
                    while ((line = reader.readLine()) != null) {
                        rawLines.add(line);
                    }
                    Point exit = null;
                    if (!rawLines.isEmpty() &&
rawLines.get(0).trim().equals("K")) {
                        String firstLine = rawLines.get(0);
                        int x = firstLine.indexOf('K');
                        exit = new Point(x, height);
                        rawLines.remove(0);
                    }
                    if (!rawLines.isEmpty() && rawLines.get(rawLines.size() -
1).trim().equals("K")) {
                        String lastLine = rawLines.get(rawLines.size() - 1);
                        int x = lastLine.indexOf('K');
```

```
                        exit = new Point(x, -1);
                        rawLines.remove(rawLines.size() - 1);
                    }
                    List<String> trimmedLines = new ArrayList<>();
                    for (String lines : rawLines) {
                        if (lines != null) {
                            String trimmed = lines.trim();
                            if (!trimmed.isEmpty()) {
                                trimmedLines.add(trimmed);
                            }
                        }
                    }
                    if (trimmedLines.size() != height) {
                        throw new IOException("Jumlah baris grid (" +
trimmedLines.size() + ") tidak sesuai dimensi height (" + height + ")");
                    } else {
                        for (int i = 0 ; i < height ; i++) {
                            String row = trimmedLines.get(i);
                            if (row.startsWith("K")) {
                                int newY = (i + height - 1);
                                if (newY >= height) {
                                    newY -= ((newY - (height - 1)) * 2);
                                }
                                exit = new Point(-1 , newY);
                                trimmedLines.set(i , row.substring(1));
                            } else if (row.endsWith("K")) {
                                int newY = (i + height - 1);
                                if (newY >= height) {
                                    newY -= ((newY - (height - 1)) * 2);
                                }
                                exit = new Point(width , newY);
                                trimmedLines.set(i , row.substring(0 ,
row.length() - 1));
                            }
                        }
                        char[][] grid = new char[height][width];
                        Board board = new Board(width , height);
                        for (int i = 0 ; i < height ; i++) {
                            String row = trimmedLines.get(i);
                            for (int j = 0 ; j < width ; j++) {
                                char c = '.';
                                if ((j < row.length())) {
                                    c = row.charAt(j);
                                }
                                grid[i][j] = c;
                                board.setCell(j , i , c);
                            }
                        }
                        List<Piece> pieces = readPieces(grid);
                        return new DataStructure(width , height , pieceCount
, exit , board , pieces);
                    }
                }
            }
        }
    }

    public static List<Piece> readPieces(char[][] grid) {
        // DESKRIPSI LOKAL
        // Melakukan pembacaan pieces pada grid yang telah diread sebelumnya.

        // KAMUS LOKAL
        // pieceCoordinates : Map From Key Character To Value List of Class
Point
        // pieces : List of Class Piece
        // coords : List of Class Point
        // grid : Matrix of Character
        // cell , type : Character
```

```
        // entry : Map Entry
        // width , height , orientation , i , j : Integer

        // ALGORITMA LOKAL
        int height = grid.length;
        int width = grid[0].length;
        Map<Character , List<Point>> pieceCoordinates = new HashMap<>();
        for (int i = 0 ; i < height ; i++) {
            for (int j = 0 ; j < width ; j++) {
                char cell = grid[i][j];
                if (!pieceCoordinates.containsKey(cell)) {
                    pieceCoordinates.put(cell , new ArrayList<>());
                }
                pieceCoordinates.get(cell).add(new Point(j , i));
            }
        }
        List<Piece> pieces = new ArrayList<>();
        for (Map.Entry<Character , List<Point>> entry :
pieceCoordinates.entrySet()) {
            char type = entry.getKey();
            List<Point> coords = entry.getValue();
            int orientation = determineOrientation(coords);
            pieces.add(new Piece(type , coords , orientation));
        }
        return pieces;
    }

    public static int determineOrientation(List<Point> coordinates) {
        // DESKRIPSI LOKAL
        // Melakukan penentuan orientasi sebuah piece berdasarkan titik-titik
koordinatnya.

        // KAMUS LOKAL
        // coordinates : List of Class Point
        // grid : Matrix of Character
        // point : Class Point
        // cell , type : Character
        // firstX , firstY : Integer
        // isHorizontal : boolean

        // ALGORITMA LOKAL
        boolean isHorizontal = true;
        int firstX = coordinates.get(0).getX();
        int firstY = coordinates.get(0).getY();
        for (Point point : coordinates) {
            if (point.getY() != firstY) {
                isHorizontal = false;
                break;
            }
            if (point.getX() != firstX) {
                break;
            }
        }
        if (isHorizontal) {
            return 0;
        } else {
            return 1;
        }
    }
}
```

9. Saver.java

```
/* Kelompok   : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1   : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1    : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB  */
/* Nama - 2   : Rafif Farras                                         */
/* NIM - 2    : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB  */
```

```java
/* Tanggal    : Rabu, 21 Mei 2025                                          */
/* Tugas      : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)              */
/* File Path  : Tucil3_13523021_13523095/src/utils/Saver.java              */
/* Deskripsi  : F11 - File Saver Utility                                   */
/* PIC F11    : K01 - 13523021 - Muhammad Raihan Nazhim Oktana             */

// Package & Import
package utils;
import algorithm.*;
import game.*;
import java.io.*;

// Class Definition & Implementation
public final class Saver {
   // DESKRIPSI
   // Public Class Saver

   // KAMUS
   // Saver : Constructor Class Saver
   // saveFile , writeBoard , writeLastBoard : Procedure

   // PRIVATE ATTRIBUTES
   // None

   private Saver() {
       // DESKRIPSI LOKAL
       // Instansiasi Constructor Class Saver

       // KAMUS LOKAL
       // None

       // ALGORITMA LOKAL
       // None
   }

   public static void saveFile(String filePath , Solution solution) throws
IOException {
       // DESKRIPSI LOKAL
       // Melakukan write solution pada file yang telah ditentukan pathnya.

       // KAMUS LOKAL
       // filePath : String
       // solution : Class Solution
       // writer : Java IO BufferedWriter
       // piece : Class Piece
       // primaryPieceSize , i , j : Integer

       // ALGORITMA LOKAL
       try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath))) {

writer.write("================================================\n");
           writer.write("INFORMATION SOLUTION RESULT :\n");
           writer.write(String.format("Algorithm    : %s\n" ,
solution.getAlgorithm()));
           writer.write(String.format("Heuristic    : %d\n" ,
solution.getHeuristicId()));
           writer.write(String.format("Step Count   : %d Step\n" ,
solution.getStepCount()));
           writer.write(String.format("Visited Node : %d Node\n" ,
solution.getNodesVisited()));
           writer.write(String.format("Time Usage   : %d ms\n" , (int)
solution.getTime()));
           if (solution.getMoves().isEmpty()) {
               writer.write("Success      : NO\n");
           } else {
               writer.write("Success      : YES\n");
           }
```

```
writer.write("================================================\n");
            if (solution.getMoves().isEmpty()) {
                writer.write("\nDisplay Board Awal & Akhir :\n");
                writeBoard(writer , solution.getPath().get(0));
            } else {
                writer.write("Display Board Awal :\n");
                writeBoard(writer , solution.getPath().get(0));
                for (int i = 0 ; i < solution.getMoves().size() ; i++) {
                    writer.write(String.format("\nMOVE %d : %s\n" , (i + 1) ,
solution.getMoves().get(i)));
                    writeBoard(writer , solution.getPath().get(i + 1));
                }
                if (!solution.getMoves().isEmpty()) {
                    GameLogic.Move last =
solution.getMoves().get(solution.getMoves().size() - 1);
                    String direction = switch (last.getDirection()) {
                        case UP    -> "UP";
                        case DOWN  -> "DOWN";
                        case LEFT  -> "LEFT";
                        case RIGHT -> "RIGHT";
                        default    -> "UNKNOWN";
                    };
                    int primaryPieceSize =
solution.getPath().get(0).getPieces().stream().filter(piece ->
piece.getType() == 'P').findFirst().orElseThrow().solveSize();
                    writer.write(String.format("\nMOVE %d : P - OUT %s (%d
STEP)\n" , solution.getMoves().size() + 1 , direction , primaryPieceSize));
                    writeLastBoard(writer ,
solution.getPath().get(solution.getMoves().size() - 1));
                }
            }

writer.write("================================================\n");
            writer.write("RECALL INFORMATION SOLUTION RESULT :\n");
            writer.write(String.format("Algorithm    : %s\n" ,
solution.getAlgorithm()));
            writer.write(String.format("Heuristic    : %d\n" ,
solution.getHeuristicId()));
            writer.write(String.format("Step Count   : %d Step\n" ,
solution.getStepCount()));
            writer.write(String.format("Visited Node : %d Node\n" ,
solution.getNodesVisited()));
            writer.write(String.format("Time Usage   : %d ms\n" , (int)
solution.getTime()));
            if (solution.getMoves().isEmpty()) {
                writer.write("Success      : NO\n");
            } else {
                writer.write("Success      : YES\n");
            }

writer.write("================================================\n");
        }
    }

    private static void writeBoard(BufferedWriter writer , DataStructure
dataStructure) throws IOException {
        // DESKRIPSI LOKAL
        // Menulis Board Puzzle Ke Writer

        // KAMUS LOKAL
        // writer : Java IO BufferedWriter
        // dataStructure : Class DataStructure
        // board : Class Board
        // exit : Class Point
        // width , height , i , j : Integer

        // ALGORITMA LOKAL
```

```
        Board board = dataStructure.getBoard();
        Point exit = dataStructure.getExit();
        int width = dataStructure.getWidth();
        int height = dataStructure.getHeight();
        if (exit.getX() == -1) {
            for (int i = 0 ; i < height ; i++) {
                if (height - 1 - exit.getY() == i) {
                    writer.write("K ");
                } else {
                    writer.write("  ");
                }
                for (int j = 0 ; j < width ; j++) {
                    writer.write(board.getCell(j , i) + " ");
                }
                writer.newLine();
            }
        } else if (exit.getY() == -1) {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    writer.write(board.getCell(j , i) + " ");
                }
                writer.newLine();
            }
            for (int j = 0 ; j < exit.getX() ; j++) {
                writer.write("  ");
            }
            writer.write("K\n");
        } else if (exit.getY() == height) {
            for (int j = 0 ; j < exit.getX() ; j++) {
                writer.write("  ");
            }
            writer.write("K\n");
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    writer.write(board.getCell(j , i) + " ");
                }
                writer.newLine();
            }
        } else {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    writer.write(board.getCell(j , i) + " ");
                }
                if (height - 1 - exit.getY() == i) {
                    writer.write("K");
                }
                writer.newLine();
            }
        }
    }

    private static void writeLastBoard(BufferedWriter writer , DataStructure
dataStructure) throws IOException {
        // DESKRIPSI LOKAL
        // Menulis Last Board Puzzle Ke Writer

        // KAMUS LOKAL
        // writer : Java IO BufferedWriter
        // dataStructure : Class DataStructure
        // piece : Class Piece
        // board : Class Board
        // exit : Class Point
        // primaryPieceSize , width , height , i , j : Integer

        // ALGORITMA LOKAL
        Board board = dataStructure.getBoard();
        Point exit = dataStructure.getExit();
        int width = dataStructure.getWidth();
```

```
        int height = dataStructure.getHeight();
        int primaryPieceSize =
dataStructure.getPieces().stream().filter(piece -> piece.getType() ==
'P').findFirst().orElseThrow().solveSize();
        if (exit.getX() == -1) {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < primaryPieceSize ; j++) {
                    if (height - 1 - exit.getY() == i) {
                        writer.write("P ");
                    } else {
                        writer.write("  ");
                    }
                }
                for (int j = 0 ; j < width ; j++) {
                    if (board.getCell(j , i) == 'P') {
                        writer.write(". ");
                    } else {
                        writer.write(board.getCell(j , i) + " ");
                    }
                }
                writer.newLine();
            }
        } else if (exit.getY() == -1) {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    if (board.getCell(j , i) == 'P') {
                        writer.write(". ");
                    } else {
                        writer.write(board.getCell(j , i) + " ");
                    }
                }
                writer.newLine();
            }
            for (int i = 0 ; i < primaryPieceSize ; i++) {
                for (int j = 0 ; j < exit.getX() ; j++) {
                    writer.write("  ");
                }
                writer.write("P\n");
            }
        } else if (exit.getY() == height) {
            for (int i = 0 ; i < primaryPieceSize ; i++) {
                for (int j = 0 ; j < exit.getX() ; j++) {
                    writer.write("  ");
                }
                writer.write("P\n");
            }
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    if (board.getCell(j , i) == 'P') {
                        writer.write(". ");
                    } else {
                        writer.write(board.getCell(j , i) + " ");
                    }
                }
                writer.newLine();
            }
        } else {
            for (int i = 0 ; i < height ; i++) {
                for (int j = 0 ; j < width ; j++) {
                    if (board.getCell(j , i) == 'P') {
                        writer.write(". ");
                    } else {
                        writer.write(board.getCell(j , i) + " ");
                    }
                }
                if (height - 1 - exit.getY() == i) {
                    for (int j = 0 ; j < primaryPieceSize ; j++) {
                        writer.write("P ");
```

```
                    }
                }
                writer.newLine();
            }
        }
    }
}
```

10. DataStructure.java

```
/* Kelompok   : Kelompok Tucil3_13523021_13523095                        */
/* Nama - 1   : Muhammad Raihan Nazhim Oktana                            */
/* NIM - 1    : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB     */
/* Nama - 2   : Rafif Farras                                             */
/* NIM - 2    : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB     */
/* Tanggal    : Rabu, 21 Mei 2025                                        */
/* Tugas      : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)           */
/* File Path  : Tucil3_13523021_13523095/src/utils/DataStructure.java    */
/* Deskripsi  : F07A - Data Structure Utility                            */
/* PIC F07A   : K01 - 13523021 - Muhammad Raihan Nazhim Oktana           */

// Package & Import
package utils;
import algorithm.*;
import java.util.*;

// Class Definition & Implementation
public class DataStructure {
    // DESKRIPSI
    // Public Class DataStructure

    // KAMUS
    // DataStructure : Constructor Class DataStructure
    // getWidth , getHeight , getPieceCount , getExit , getBoard , getPieces
: Procedure
    // setWidth , setHeight , setPieceCount , setExit , setBoard , setPieces
: Procedure
    // displayBoard , displayLastBoard , displayDataStructure : Procedure

    // PRIVATE ATTRIBUTES
    private int width;
    private int height;
    private int pieceCount;
    private Point exit;
    private Board board;
    private List<Piece> pieces;

    public DataStructure(int width , int height , int pieceCount , Point exit
, Board board , List<Piece> pieces) {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class DataStructure

        // KAMUS LOKAL
        // pieces : List Of Class Piece
        // board : Class Board
        // exit : Class Point
        // width , height , pieceCount : Integer

        // ALGORITMA LOKAL
        this.width = width;
        this.height = height;
        this.pieceCount = pieceCount;
        this.exit = exit;
        this.board = board;
        this.pieces = pieces;
    }

    public int getWidth() {
        // DESKRIPSI LOKAL
```

```
    // Getter Width

    // KAMUS LOKAL
    // width : Integer

    // ALGORITMA LOKAL
    return this.width;
}

public int getHeight() {
    // DESKRIPSI LOKAL
    // Getter Height

    // KAMUS LOKAL
    // height : Integer

    // ALGORITMA LOKAL
    return this.height;
}

public int getPieceCount() {
    // DESKRIPSI LOKAL
    // Getter Piece Count

    // KAMUS LOKAL
    // pieceCount : Integer

    // ALGORITMA LOKAL
    return this.pieceCount;
}

public Point getExit() {
    // DESKRIPSI LOKAL
    // Getter Exit

    // KAMUS LOKAL
    // exit : Class Point

    // ALGORITMA LOKAL
    return this.exit;
}

public Board getBoard() {
    // DESKRIPSI LOKAL
    // Getter Board

    // KAMUS LOKAL
    // board : Class Board

    // ALGORITMA LOKAL
    return this.board;
}

public List<Piece> getPieces() {
    // DESKRIPSI LOKAL
    // Getter Pieces

    // KAMUS LOKAL
    // pieces : List Of Class Piece

    // ALGORITMA LOKAL
    return this.pieces;
}

public void setWidth(int width) {
    // DESKRIPSI LOKAL
    // Setter Width
```

```
        // KAMUS LOKAL
        // width : Integer

        // ALGORITMA LOKAL
        this.width = width;
    }

    public void setHeight(int height) {
        // DESKRIPSI LOKAL
        // Setter Height

        // KAMUS LOKAL
        // height : Integer

        // ALGORITMA LOKAL
        this.height = height;
    }

    public void setPieceCount(int pieceCount) {
        // DESKRIPSI LOKAL
        // Setter Piece Count

        // KAMUS LOKAL
        // pieceCount : Integer

        // ALGORITMA LOKAL
        this.pieceCount = pieceCount;
    }

    public void setExit(Point exit) {
        // DESKRIPSI LOKAL
        // Setter Exit

        // KAMUS LOKAL
        // exit : Class Point

        // ALGORITMA LOKAL
        this.exit = exit;
    }

    public void setBoard(Board board) {
        // DESKRIPSI LOKAL
        // Setter Board

        // KAMUS LOKAL
        // board : Class Board

        // ALGORITMA LOKAL
        this.board = board;
    }

    public void setPieces(List<Piece> pieces) {
        // DESKRIPSI LOKAL
        // Setter Pieces

        // KAMUS LOKAL
        // pieces : List Of Class Piece

        // ALGORITMA LOKAL
        this.pieces = pieces;
    }

    public void displayBoard() {
        // DESKRIPSI LOKAL
        // Display Board From Data Structure To CLI Terminal

        // KAMUS LOKAL
        // board : Class Board
```

```
        // exit : Class Point
        // width , height , i , j : Integer

        // ALGORITMA LOKAL
        if (this.exit.getX() == -1) {
            for (int i = 0 ; i < this.height ; i++) {
                if (this.height - 1 - this.exit.getY() == i) {
                    System.out.print("K ");
                } else {
                    System.out.print("  ");
                }
                for (int j = 0 ; j < this.width ; j++) {
                    System.out.print(this.board.getCell(j , i) + " ");
                }
                System.out.println();
            }
        } else if (this.exit.getY() == -1) {
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.width ; j++) {
                    System.out.print(this.board.getCell(j , i) + " ");
                }
                System.out.println();
            }
            for (int i = 0 ; i < this.exit.getX() ; i++) {
                System.out.print("  ");
            }
            System.out.println("K");
        } else if (this.exit.getY() == this.height) {
            for (int i = 0 ; i < this.exit.getX() ; i++) {
                System.out.print("  ");
            }
            System.out.println("K");
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.width ; j++) {
                    System.out.print(this.board.getCell(j , i) + " ");
                }
                System.out.println();
            }
        } else {
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.width ; j++) {
                    System.out.print(this.board.getCell(j , i) + " ");
                }
                if (this.height - 1 - this.exit.getY() == i) {
                    System.out.print("K");
                }
                System.out.println();
            }
        }
    }

    public void displayLastBoard() {
        // DESKRIPSI LOKAL
        // Display Last Board From Data Structure To CLI Terminal

        // KAMUS LOKAL
        // board : Class Board
        // exit : Class Point
        // width , height , i , j : Integer

        // ALGORITMA LOKAL
        if (this.exit.getX() == -1) {
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.pieces.stream().filter(pc ->
pc.getType() == 'P').findFirst().orElseThrow().solveSize() ; j++) {
                    if (this.height - 1 - this.exit.getY() == i) {
                        System.out.print(Solution.ANSI_BACKGROUND_GREEN +
Solution.ANSI_BOLD + Solution.ANSI_CYAN + "P " + Solution.ANSI_RESET);
```

```
                    } else {
                        System.out.print("  ");
                    }
                }
                for (int j = 0 ; j < this.width ; j++) {
                    if (this.board.getCell(j , i) == 'P') {
                        System.out.print(Solution.ANSI_BACKGROUND_RED + "." +
Solution.ANSI_RESET + " ");
                    } else {
                        System.out.print(this.board.getCell(j , i) + " ");
                    }
                }
                System.out.println();
            }
        } else if (this.exit.getY() == -1) {
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.width ; j++) {
                    if (this.board.getCell(j , i) == 'P') {
                        System.out.print(Solution.ANSI_BACKGROUND_RED + "." +
Solution.ANSI_RESET + " ");
                    } else {
                        System.out.print(this.board.getCell(j , i) + " ");
                    }
                }
                System.out.println();
            }
            for (int i = 0 ; i < this.pieces.stream().filter(pc ->
pc.getType() == 'P').findFirst().orElseThrow().solveSize() ; i++) {
                for (int j = 0 ; j < this.exit.getX() ; j++) {
                    System.out.print("  ");
                }
                System.out.println(Solution.ANSI_BACKGROUND_GREEN +
Solution.ANSI_BOLD + Solution.ANSI_CYAN + "P " + Solution.ANSI_RESET);
            }
        } else if (this.exit.getY() == this.height) {
            for (int i = 0 ; i < this.pieces.stream().filter(pc ->
pc.getType() == 'P').findFirst().orElseThrow().solveSize() ; i++) {
                for (int j = 0 ; j < this.exit.getX() ; j++) {
                    System.out.print("  ");
                }
                System.out.println(Solution.ANSI_BACKGROUND_GREEN +
Solution.ANSI_BOLD + Solution.ANSI_CYAN + "P " + Solution.ANSI_RESET);
            }
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.width ; j++) {
                    if (this.board.getCell(j , i) == 'P') {
                        System.out.print(Solution.ANSI_BACKGROUND_RED + "." +
Solution.ANSI_RESET + " ");
                    } else {
                        System.out.print(this.board.getCell(j , i) + " ");
                    }
                }
                System.out.println();
            }
        } else {
            for (int i = 0 ; i < this.height ; i++) {
                for (int j = 0 ; j < this.width ; j++) {
                    if (this.board.getCell(j , i) == 'P') {
                        System.out.print(Solution.ANSI_BACKGROUND_RED + "." +
Solution.ANSI_RESET + " ");
                    } else {
                        System.out.print(this.board.getCell(j , i) + " ");
                    }
                }
                if (this.height - 1 - this.exit.getY() == i) {
                    for (int j = 0 ; j < this.pieces.stream().filter(pc ->
pc.getType() == 'P').findFirst().orElseThrow().solveSize() ; j++) {
```

```
                        System.out.print(Solution.ANSI_BACKGROUND_GREEN +
Solution.ANSI_BOLD + Solution.ANSI_CYAN + "P " + Solution.ANSI_RESET);
                    }
                }
                System.out.println();
            }
        }
        System.out.println("\nNotes :");
        System.out.println(Solution.ANSI_BACKGROUND_RED + " " +
Solution.ANSI_RESET + "  : Old Position Piece P");
        System.out.println(Solution.ANSI_BACKGROUND_GREEN + "P " +
Solution.ANSI_RESET + " : New Position Piece P");
        System.out.println(Solution.ANSI_BOLD + Solution.ANSI_CYAN + "P" +
Solution.ANSI_RESET + "  : Primary Piece");
        System.out.println(Solution.ANSI_BOLD + Solution.ANSI_PURPLE + "K" +
Solution.ANSI_RESET + "  : Exit Position");
    }

    public void displayDataStructure() {
        // DESKRIPSI LOKAL
        // Display Data Structure CLI Terminal

        // KAMUS LOKAL
        // pieces : List Of Class Piece
        // piece : Class Piece
        // board : Class Board
        // exit : Class Point
        // width , height , pieceCount , i , j : Integer

        // ALGORITMA LOKAL
        System.out.println("Width Size    : " + this.width);
        System.out.println("Height Size   : " + this.height);
        System.out.println("Piece Count   : " + this.pieceCount);
        Point exitPoint = new Point(this.exit.getX() , this.exit.getY());
        if (this.exit.getY() == -1) {
            exitPoint.setY(this.height);
        } else if (this.exit.getY() == this.height) {
            exitPoint.setY(-1);
        } else {
            exitPoint.setY(this.height - 1 - this.exit.getY());
        }
        System.out.println("Exit Position : " + exitPoint);
        System.out.println("Pieces :");
        for (Piece piece : this.pieces) {
            if (piece.getType() == '.') {
                System.out.println("  Type : " + piece.getType() + " ,
Coordinates : [Other]" + " , Orientation : Point Dot Blank Space (-)");
            } else {
                if (piece.getOrientation() == 0) {
                    System.out.println("  Type : " + piece.getType() + " ,
Coordinates : " + piece.getCoordinates() + " , Orientation : Horizontal
(0)");
                } else {
                    System.out.println("  Type : " + piece.getType() + " ,
Coordinates : " + piece.getCoordinates() + " , Orientation : Vertical (1)");
                }
            }
        }
        System.out.println("Puzzle Board :");
        this.displayBoard();
    }
}
```

11. Board.java

```
/* Kelompok   : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1   : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1    : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB */
/* Nama - 2   : Rafif Farras                                         */
```

```java
/* NIM - 2    : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB   */
/* Tanggal    : Rabu, 21 Mei 2025                                      */
/* Tugas      : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)         */
/* File Path  : Tucil3_13523021_13523095/src/utils/Board.java          */
/* Deskripsi  : F07D - Data Structure Board Utility                    */
/* PIC F07D   : K02 - 13523095 - Rafif Farras                          */

// Package & Import
package utils;

// Class Definition & Implementation
public class Board {
   // DESKRIPSI
   // Public Class Board

   // KAMUS
   // Board : Constructor Class Board
   // getGrid , getCell , getWidth , getHeight , setGrid , setCell ,
setWidth , setHeight : Procedure

   // PRIVATE ATTRIBUTES
   private char[][] grid;
   private int width;
   private int height;

   public Board(int width , int height) {
       // DESKRIPSI LOKAL
       // Instansiasi Constructor Class Board 1

       // KAMUS LOKAL
       // grid : Matrix of Character
       // width , height : Integer

       // ALGORITMA LOKAL
       this.width = width;
       this.height = height;
       this.grid = new char[height][width];
   }

   public Board(char[][] grid) {
       // DESKRIPSI LOKAL
       // Instansiasi Constructor Class Board 2

       // KAMUS LOKAL
       // grid : Matrix of Character
       // width , height : Integer

       // ALGORITMA LOKAL
       this.grid = grid;
       this.height = grid.length;
       this.width = grid[0].length;
   }

   public Board(Board board) {
       // DESKRIPSI LOKAL
       // Instansiasi Constructor Class Board 3

       // KAMUS LOKAL
       // grid : Matrix of Character
       // width , height : Integer

       // ALGORITMA LOKAL
       this.width = board.getWidth();
       this.height = board.getHeight();
       this.grid = new char[this.height][this.width];
       for (int i = 0 ; i < this.height ; i++) {
           System.arraycopy(board.getGrid()[i] , 0 , this.grid[i] , 0 ,
width);
```

```
        }
    }

    public char[][] getGrid() {
        // DESKRIPSI LOKAL
        // Getter Grid

        // KAMUS LOKAL
        // grid : Matrix of Character

        // ALGORITMA LOKAL
        return this.grid;
    }

    public char getCell(int x , int y) {
        // DESKRIPSI LOKAL
        // Getter Cell

        // KAMUS LOKAL
        // grid : Matrix of Character
        // x , y : Integer

        // ALGORITMA LOKAL
        return this.grid[y][x];
    }

    public int getWidth() {
        // DESKRIPSI LOKAL
        // Getter Width

        // KAMUS LOKAL
        // width : Integer

        // ALGORITMA LOKAL
        return this.width;
    }

    public int getHeight() {
        // DESKRIPSI LOKAL
        // Getter Height

        // KAMUS LOKAL
        // height : Integer

        // ALGORITMA LOKAL
        return this.height;
    }

    public void setGrid(char[][] grid) {
        // DESKRIPSI LOKAL
        // Setter Grid

        // KAMUS LOKAL
        // grid : Matrix of Character

        // ALGORITMA LOKAL
        this.grid = grid;
    }

    public void setCell(int x , int y , char value) {
        // DESKRIPSI LOKAL
        // Setter Cell

        // KAMUS LOKAL
        // grid : Matrix of Character
        // x , y : Integer
        // value : Character
```

```
        // ALGORITMA LOKAL
        this.grid[y][x] = value;
    }

    public void setWidth(int width) {
        // DESKRIPSI LOKAL
        // Getter Width

        // KAMUS LOKAL
        // width : Integer

        // ALGORITMA LOKAL
        this.width = width;
    }

    public void setHeight(int height) {
        // DESKRIPSI LOKAL
        // Getter Height

        // KAMUS LOKAL
        // height : Integer

        // ALGORITMA LOKAL
        this.height = height;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int y = 0 ; y < this.height ; y++) {
            for (int x = 0 ; x < this.width ; x++) {
                sb.append(grid[y][x]).append(' ');
            }
            sb.append('\n');
        }
        return sb.toString();
    }
}
```

12. Piece.java

```
/* Kelompok    : Kelompok Tucil3_13523021_13523095                    */
/* Nama - 1    : Muhammad Raihan Nazhim Oktana                        */
/* NIM - 1     : K01 - 13523021 - Teknik Informatika (IF-Ganesha) ITB */
/* Nama - 2    : Rafif Farras                                         */
/* NIM - 2     : K02 - 13523095 - Teknik Informatika (IF-Ganesha) ITB */
/* Tanggal     : Rabu, 21 Mei 2025                                    */
/* Tugas       : Tugas Kecil 3 - Strategi Algoritma (IF2211-24)       */
/* File Path   : Tucil3_13523021_13523095/src/utils/Piece.java        */
/* Deskripsi   : F07C - Data Structure Piece Utility                  */
/* PIC F07C    : K02 - 13523095 - Rafif Farras                        */

// Package & Import
package utils;
import java.util.List;

// Class Definition & Implementation
public class Piece {
    // DESKRIPSI
    // Public Class Piece

    // KAMUS
    // Piece : Constructor Class Piece
    // getType , getCoordinates , getOrientation , setType , setCoordinates ,
setOrientation : Procedure
    // solveSize : Function

    // PRIVATE ATTRIBUTES
    private char type;
```

```
    private List<Point> coordinates;
    private int orientation;

    public Piece(char type , List<Point> coordinates , int orientation) {
        // DESKRIPSI LOKAL
        // Instansiasi Constructor Class Piece

        // KAMUS LOKAL
        // type : Character
        // coordinates : List Of Class Point
        // orientation : Integer

        // ALGORITMA LOKAL
        this.type = type;
        this.coordinates = coordinates;
        this.orientation = orientation;
    }

    public char getType() {
        // DESKRIPSI LOKAL
        // Getter Type

        // KAMUS LOKAL
        // type : Character

        // ALGORITMA LOKAL
        return this.type;
    }

    public List<Point> getCoordinates() {
        // DESKRIPSI LOKAL
        // Getter Coordinates

        // KAMUS LOKAL
        // coordinates : List Of Class Point

        // ALGORITMA LOKAL
        return this.coordinates;
    }

    public int getOrientation() {
        // DESKRIPSI LOKAL
        // Getter Orientation

        // KAMUS LOKAL
        // orientation : Integer

        // ALGORITMA LOKAL
        return this.orientation;
    }

    public void setType(char type) {
        // DESKRIPSI LOKAL
        // Setter Type

        // KAMUS LOKAL
        // type : Character

        // ALGORITMA LOKAL
        this.type = type;
    }

    public void setCoordinates(List<Point> coordinates) {
        // DESKRIPSI LOKAL
        // Setter Coordinates

        // KAMUS LOKAL
        // coordinates : List Of Class Point
```

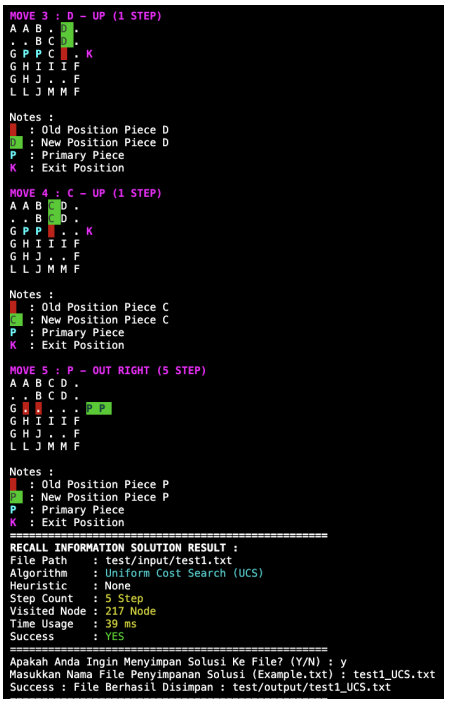
```
        // ALGORITMA LOKAL
        this.coordinates = coordinates;
    }

    public void setOrientation(int orientation) {
        // DESKRIPSI LOKAL
        // Setter Orientation

        // KAMUS LOKAL
        // orientation : Integer

        // ALGORITMA LOKAL
        this.orientation = orientation;
    }

    public int solveSize() {
        // DESKRIPSI LOKAL
        // Menghitung Ukuran Sebuah Piece

        // KAMUS LOKAL
        // rig , lft , top , bot : Integer

        // ALGORITMA LOKAL
        int rig =
this.getCoordinates().stream().mapToInt(Point::getX).max().orElseThrow();
        int lft =
this.getCoordinates().stream().mapToInt(Point::getX).min().orElseThrow();
        int top =
this.getCoordinates().stream().mapToInt(Point::getY).min().orElseThrow();
        int bot =
this.getCoordinates().stream().mapToInt(Point::getY).max().orElseThrow();
        if (this.getOrientation() == 0) {
            return Math.abs(rig - lft) + 1;
        } else {
            return Math.abs(bot - top) + 1;
        }
    }
}
```

BAB 5
UJI COBA PROGRAM

<table>
<tr><th colspan="2">Test Case #1 Metode UCS</th></tr>
<tr><td>Input</td><td>

```
6 6
11
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.
```

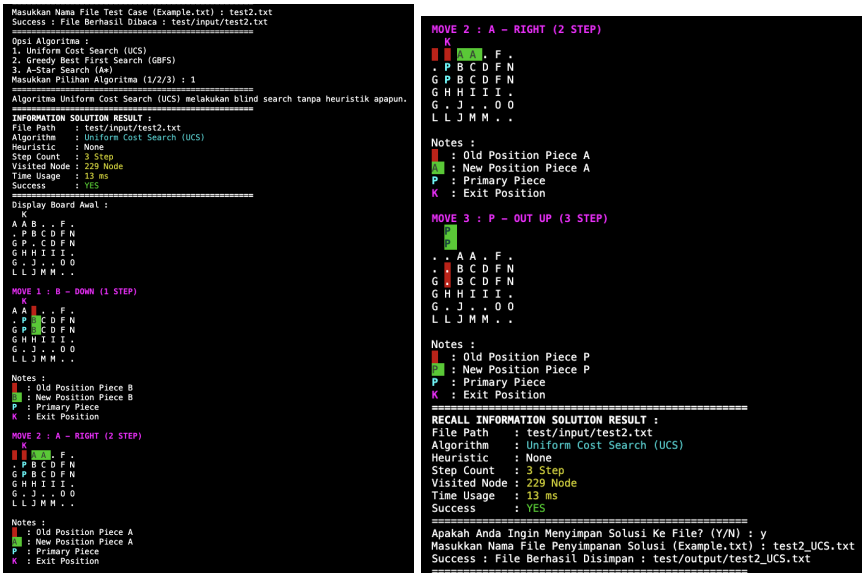
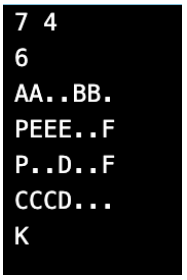
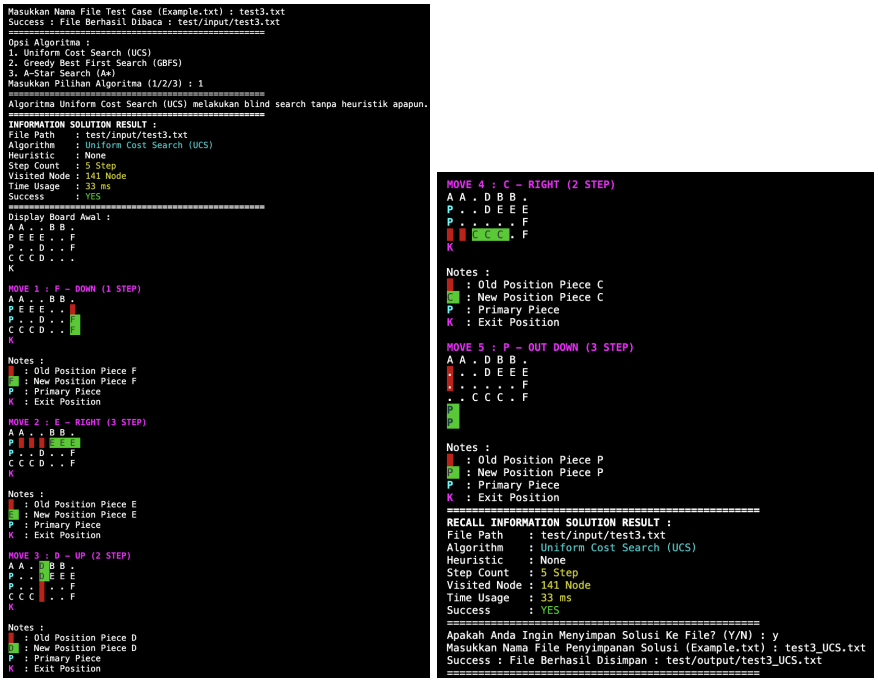
</td></tr>
<tr><td>Output</td><td>

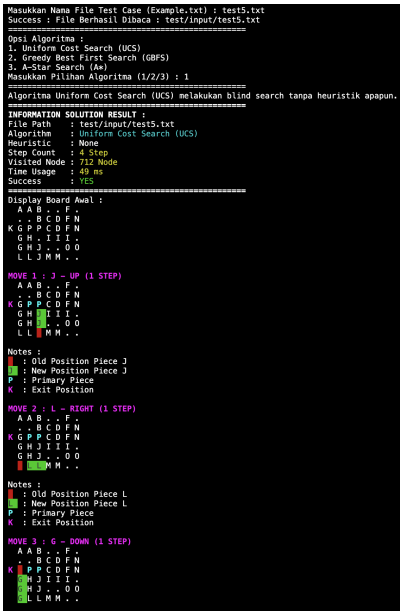
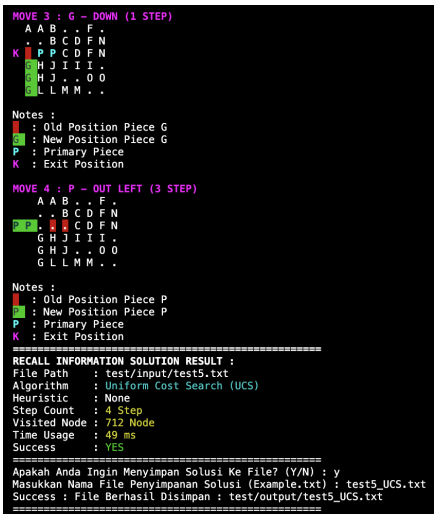


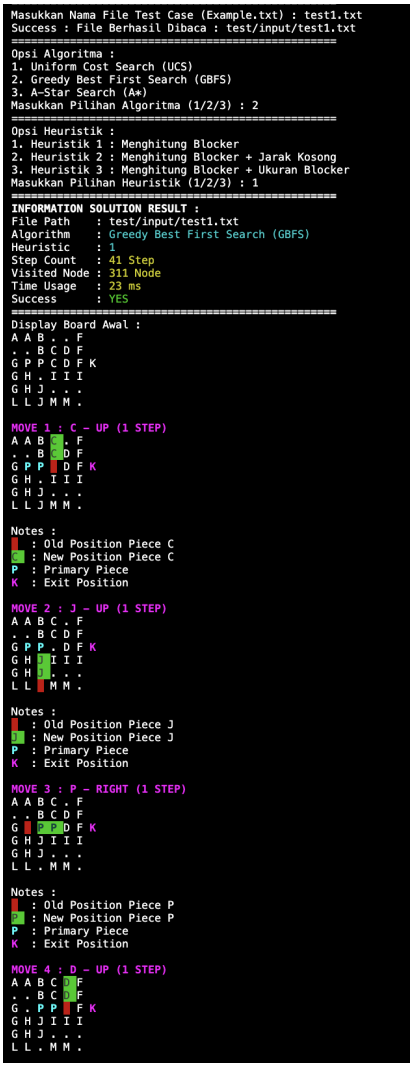
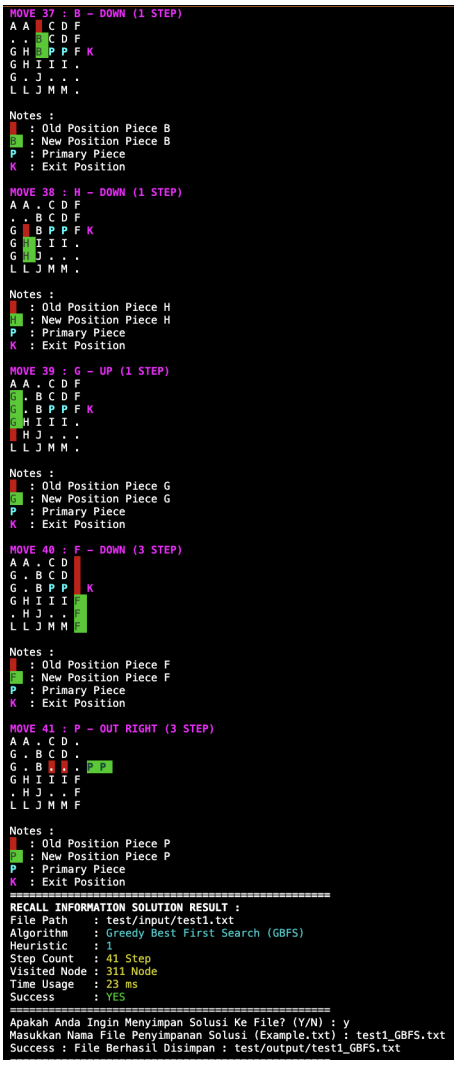
</td></tr>
<tr><th colspan="2">Test Case #2 Metode UCS</th></tr>
<tr><td>Input</td><td>

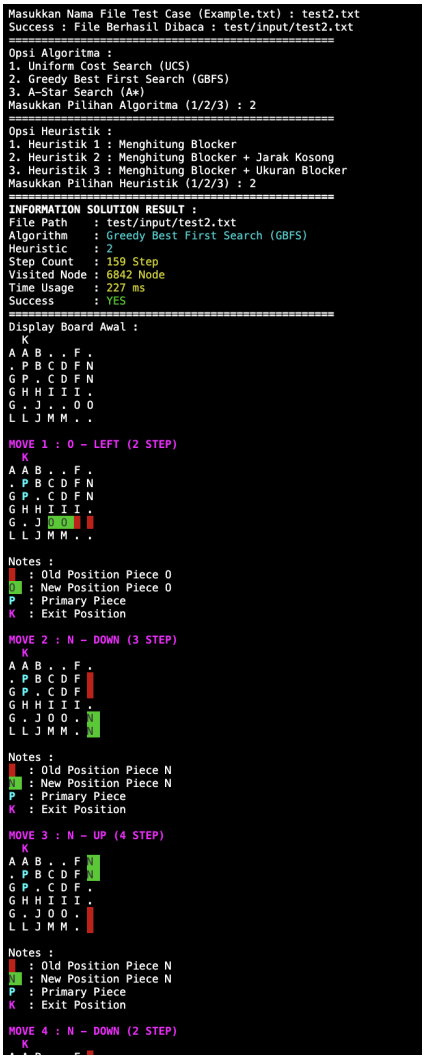
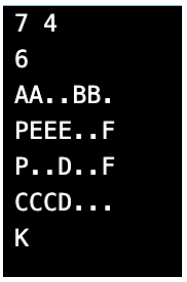
```
7 6
13
|K
AAB..F.
.PBCDFN
GP.CDFN
GHHIII.
G.J..OO
LLJMM..
```

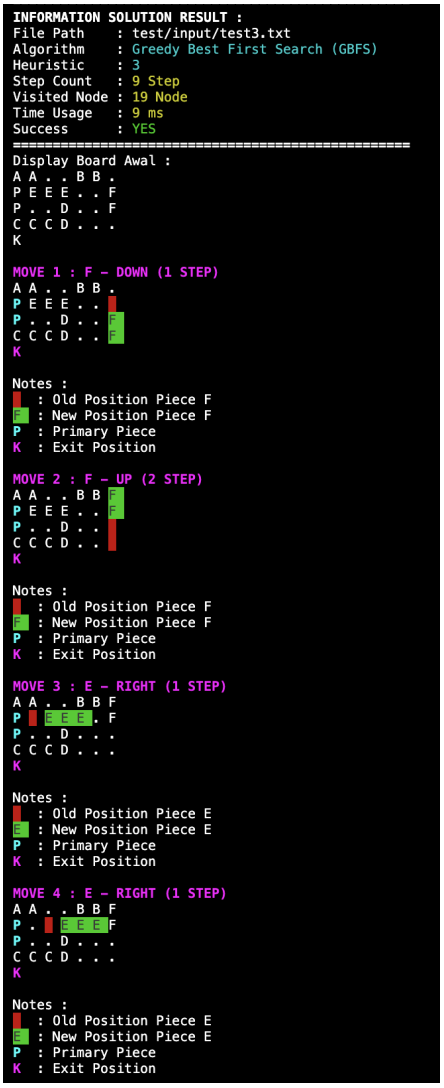
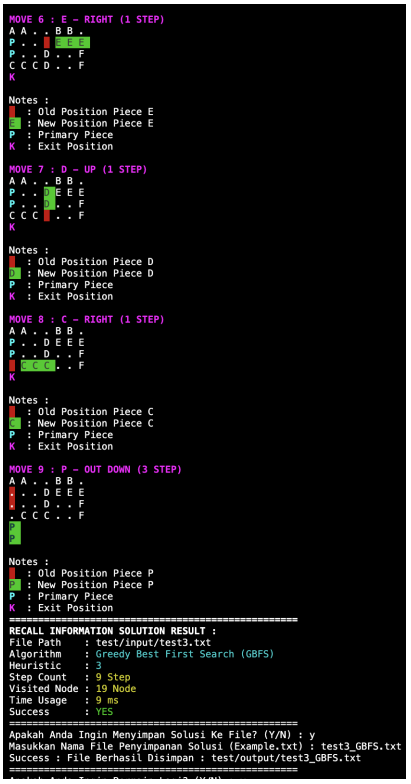
</td></tr>
</table>

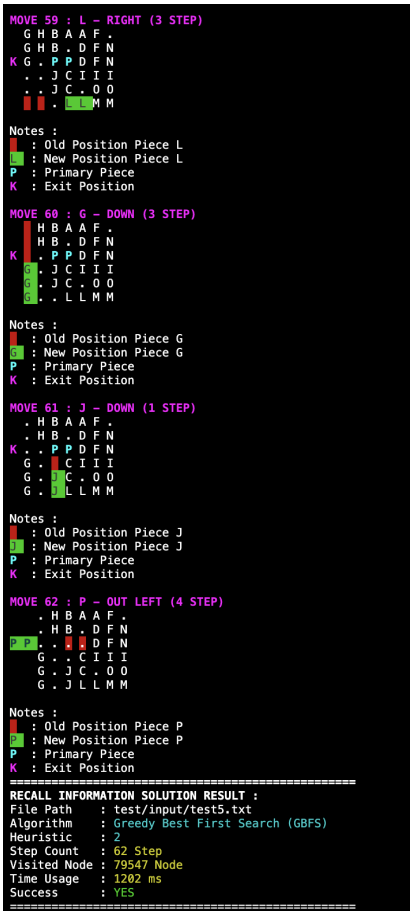
Output	
Test Case #3 Metode UCS	
Input	
Output	
Test Case #4 Metode UCS	

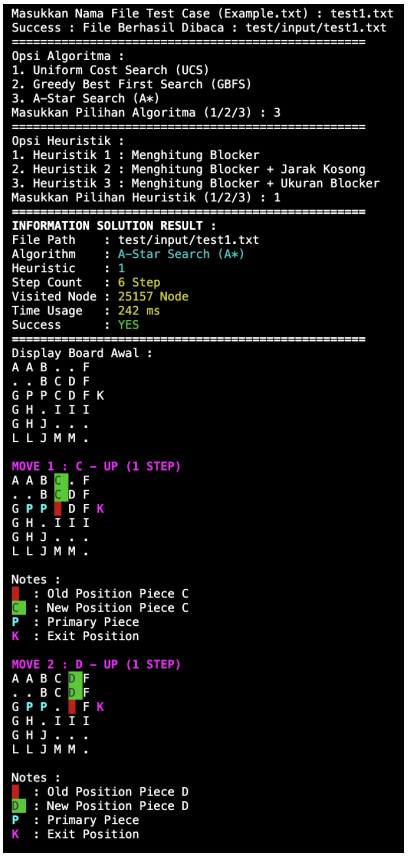
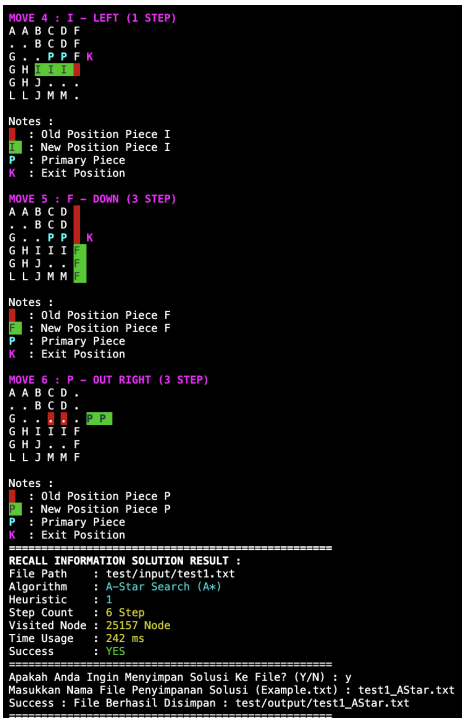
Input	
Output	 
Test Case #1 Metode GBFS	
Input	

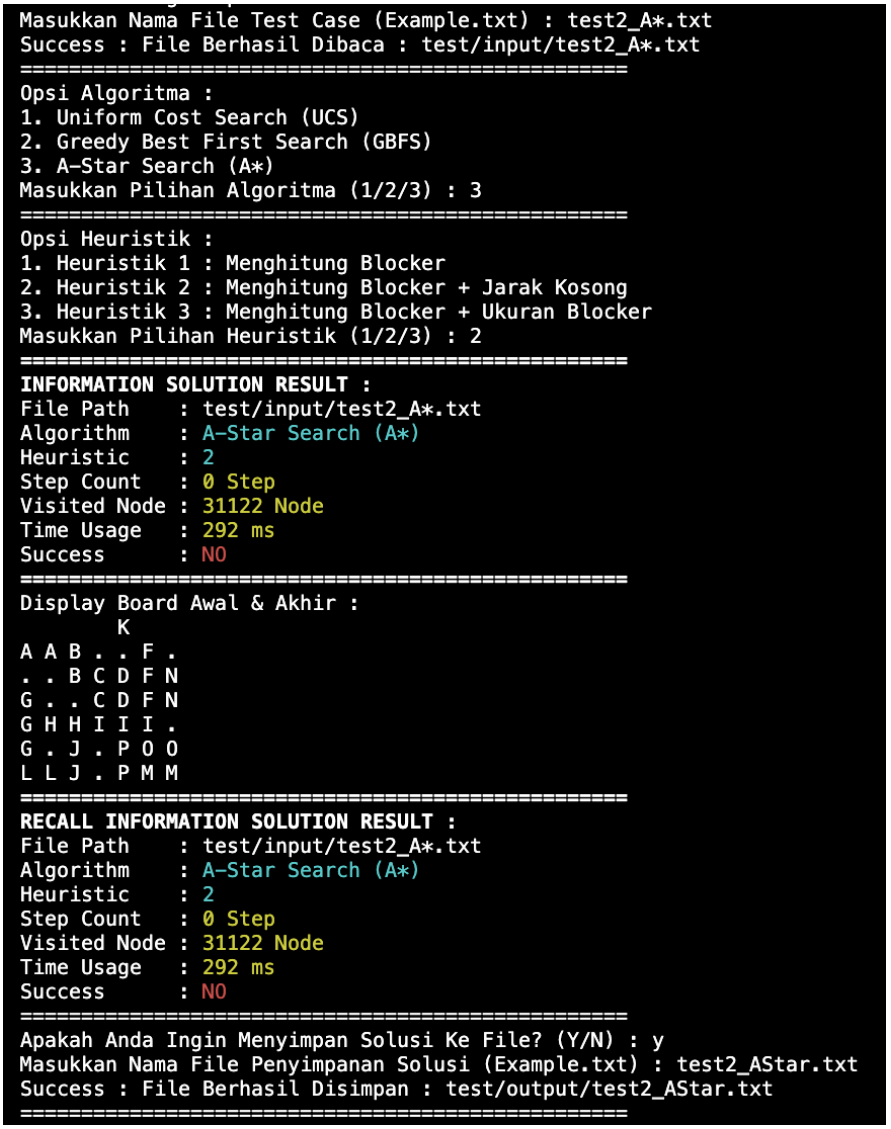
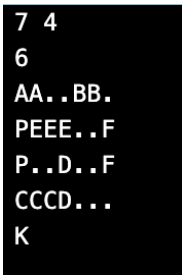
Output	 
Test Case #2 Metode GBFS	
Input	

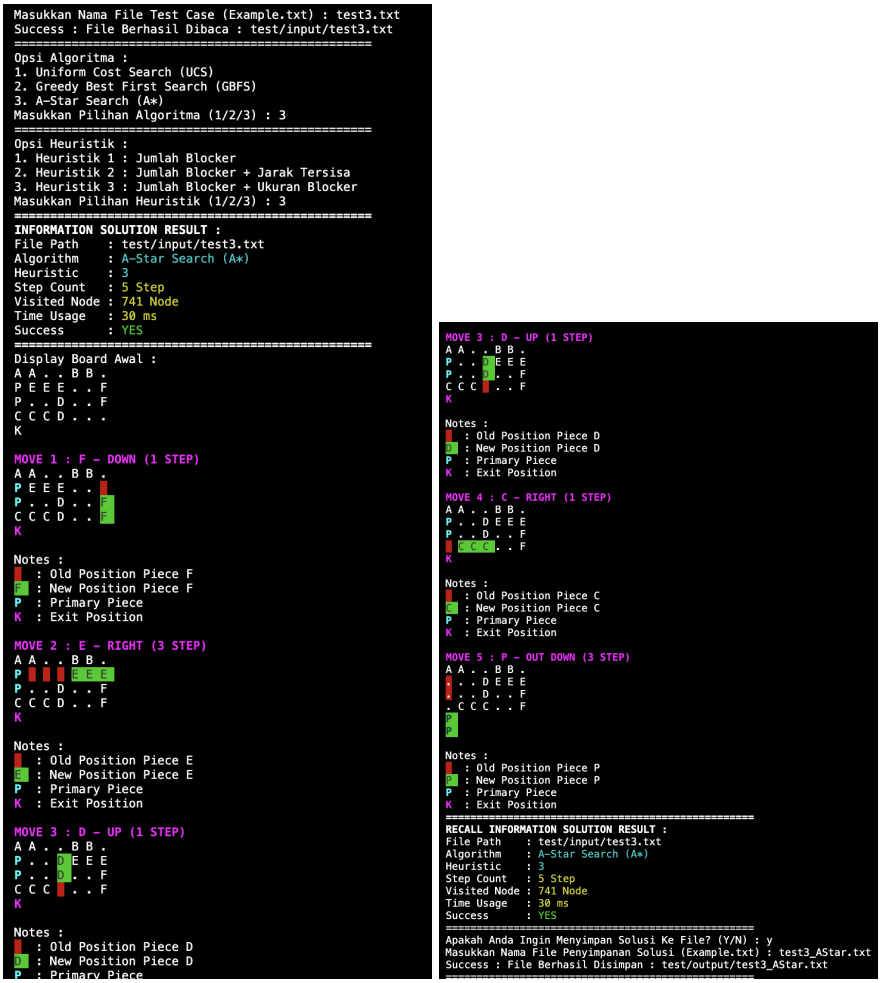
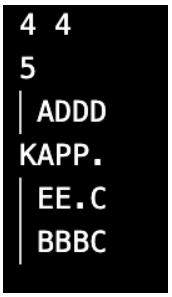
Output	
Test Case #3 Metode GBFS	
Input	

Output	 
Test Case #4 Metode GBFS	
Input	

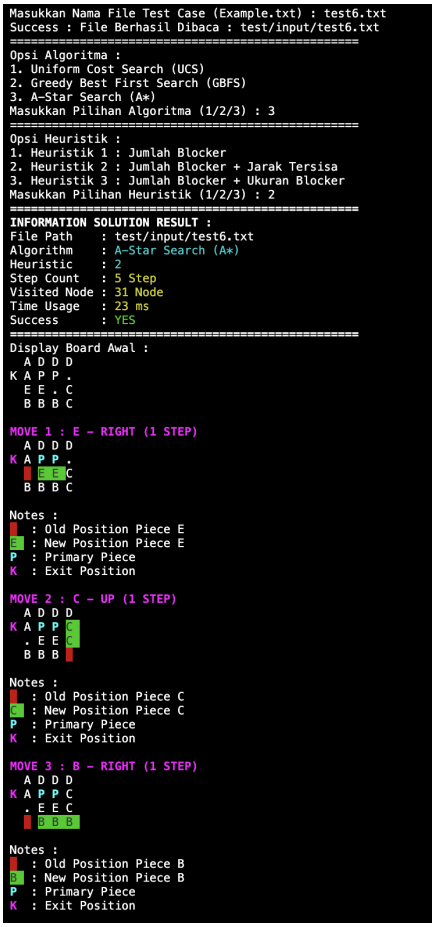
Output	 
Test Case #1 Metode A*	
Input	

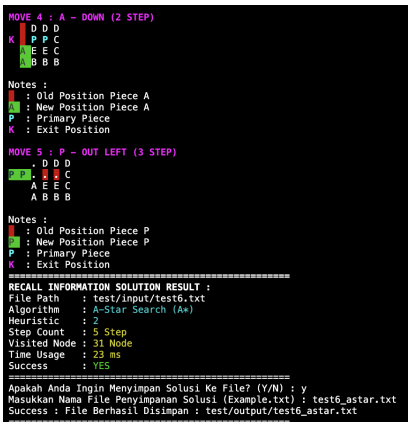
Output	 
Test Case #2 Metode A*	
Input	

Output	
Test Case #3 Metode A*	
Input	

Output	
Test Case #4 Metode A*	
Input	

Output


Masukkan Nama File Test Case (Example.txt) : test6.txt
Success : File Berhasil Dibaca : test/input/test6.txt
Opsi Algoritma :
1. Uniform Cost Search (UCS)
2. Greedy Best First Search (GBFS)
3. A-Star Search (A*)
Masukkan Pilihan Algoritma (1/2/3) : 3
Opsi Heuristik :
1. Heuristik 1 : Jumlah Blocker
2. Heuristik 2 : Jumlah Blocker + Jarak Tersisa
3. Heuristik 3 : Jumlah Blocker + Ukuran Blocker
Masukkan Pilihan Heuristik (1/2/3) : 2
INFORMATION SOLUTION RESULT :
File Path : test/input/test6.txt
Algorithm : A-Star Search (A*)
Heuristic : 2
Step Count : 5 Step
Visited Node : 31 Node
Time Usage : 23 ms
Success : YES
Display Board Awal :
A D D D
K A P P .
E E . C
B B B C
MOVE 1 : E - RIGHT (1 STEP)
A D D D
K A P P .
. E E C
B B B C
Notes :
. : Old Position Piece E
E : New Position Piece E
P : Primary Piece
K : Exit Position
MOVE 2 : C - UP (1 STEP)
A D D D
K A P P C
. E E .
B B B .
Notes :
. : Old Position Piece C
C : New Position Piece C
P : Primary Piece
K : Exit Position
MOVE 3 : B - RIGHT (1 STEP)
A D D D
K A P P C
. E E C
. B B B
Notes :
. : Old Position Piece B
B : New Position Piece B
P : Primary Piece
K : Exit Position


MOVE 4 : A - DOWN (2 STEP)
. D D D
K . P P C
A E E C
A B B B
Notes :
. : Old Position Piece A
A : New Position Piece A
P : Primary Piece
K : Exit Position
MOVE 5 : P - OUT LEFT (3 STEP)
. D D D
P P . . C
A E E C
A B B B
Notes :
. : Old Position Piece P
P : New Position Piece P
P : Primary Piece
K : Exit Position
RECALL INFORMATION SOLUTION RESULT :
File Path : test/input/test6.txt
Algorithm : A-Star Search (A*)
Heuristic : 2
Step Count : 5 Step
Visited Node : 31 Node
Time Usage : 23 ms
Success : YES
Apakah Anda Ingin Menyimpan Solusi Ke File? (Y/N) : y
Masukkan Nama File Penyimpanan Solusi (Example.txt) : test6_astar.txt
Success : File Berhasil Disimpan : test/output/test6_astar.txt

BAB 6
ANALISIS PERCOBAAN

Dengan benchmarking pada test case 1 (dikarenakan menggunakan input yang sama) setiap algoritma, didapatkan tabel sebagai berikut:

Algoritma	Step	Node Visited	Time
UCS	9	317	39ms
GBFS	41	311	33ms
A*	6	25.157	242ms

Uniform Cost Search (UCS) menunjukkan hasil yang seimbang antara optimalitas dan efisiensi. UCS menemukan solusi dalam 9 langkah, dengan 317 node yang dikunjungi, dan waktu eksekusi 39 milidetik. UCS tidak menggunakan heuristik sehingga mencari solusi dengan menyusuri semua kemungkinan berdasarkan biaya nyata dari awal, namun ternyata cukup efisien pada kasus ini karena ruang pencarian tidak terlalu besar. Hasil ini juga menunjukkan bahwa UCS dapat memberikan solusi yang lebih pendek daripada GBFS, dan jauh lebih cepat daripada A* pada kasus ini.

Greedy Best First Search (GBFS) berhasil mencapai solusi dengan waktu tercepat yaitu 33 milidetik dan hanya mengunjungi 311 node, namun solusi yang dihasilkan membutuhkan 41 langkah, yang berarti tidak optimal. Hal ini mencerminkan kelemahan utama GBFS, yaitu GBFS hanya mengandalkan estimasi heuristik ke goal (h(n)) tanpa memperhatikan langkah yang sudah ditempuh (g(n)), sehingga meskipun terlihat cepat, hasilnya sering kali tidak efisien dari segi jumlah langkah. Pada permainan seperti Rush Hour, di mana solusi optimal penting, pendekatan ini kurang cocok.

Sementara itu, A-Star (A*) memberikan solusi paling optimal dengan hanya 6 langkah, namun harus diimbangi dengan tingginya jumlah node yang dikunjungi yaitu 25.157 node, dan waktu eksekusi mencapai 242 milidetik. Ini terjadi karena meskipun A* memprioritaskan jalur terbaik secara teoritis (f(n) = g(n) + h(n)), pada kasus ini sepertinya heuristik yang kami gunakan masih belum cukup admissible, sehingga banyak node harus diekspansi untuk memastikan solusi optimal. Tetapi, A* tetap unggul dalam kualitas solusi, namun kurang efisien dalam waktu jika dibandingkan UCS pada test case ini.

BAB 7
IMPLEMENTASI BONUS

7.1. 3 Heuristic yang Digunakan

Heuristic 1 menghitung jumlah piece yang menghalangi jalur primary piece menuju keluar, dengan asumsi bahwa setiap piece penghalang menambah beban cost. Ditambah 1 sebagai nilai dasar untuk memperhitungkan langkah minimal yang diperlukan.

Heuristic 2 menggabungkan jumlah piece yang menghalangi primary piece dengan jarak langsung (horizontal atau vertikal) dari ujung primary piece ke posisi exit, sehingga memperkirakan sisa langkah yang diperlukan agar primary piece bisa keluar.

Heuristic 3 memperhitungkan jumlah piece yang menghalangi primary piece dan juga ukuran (jumlah sel) setiap piece penghalang tersebut, sehingga memberikan estimasi cost yang lebih detil berdasarkan besar penghalang.

Implementasi lengkap bisa dilihat pada bab 4 laporan ini.

LAMPIRAN

Tautan Repository: https://github.com/RNXFreeze/Tucil3_13523021_13523095

NO	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✅	
2	Program berhasil dijalankan	✅	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✅	
4	Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	✅	
5	[Bonus] Implementasi algoritma pathfinding alternatif		✅
6	[Bonus] Implementasi 2 atau lebih heuristik alternatif	✅	
7	[Bonus] Program memiliki GUI		✅
8	Program dan laporan dibuat (kelompok) sendiri	✅	