

IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Laporan Tugas Kecil

Disusun untuk memenuhi tugas kecil mata kuliah IF2211 Strategi Algoritma pada Semester II
Tahun Akademik 2024/2025



Oleh :

Angelina Efrina Prahastaputri 13523060

Sebastian Hung Yansen 13523070

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	3
1.1 Algoritma Pathfinding.....	3
1.1.1. Algoritma Uniform Cost Search (UCS).....	3
1.1.2. Algoritma Greedy Best First Search.....	3
1.1.3. Algoritma A*.....	3
1.2 Permainan Puzzle Rush Hour.....	4
Ilustrasi kasus:.....	5
2.1. Langkah Penyelesaian Permainan Puzzle Rush Hour Menggunakan Algoritma Pathfinding.....	7
2.1.1. Penyelesaian Menggunakan Algoritma Uniform Cost Search (UCS).....	7
2.1.2. Penyelesaian Menggunakan Algoritma Greedy Best First Search.....	7
2.1.3. Penyelesaian Menggunakan Algoritma A*.....	8
2.2. Analisis Algoritma Pathfinding.....	8
BAB III.....	10
3.1. Spesifikasi Teknis Program.....	10
3.1.1. Struktur Repository.....	10
3.1.2. Source Code.....	11
BAB IV.....	29
4.1. Analisis Kompleksitas Algoritma Program.....	29
4.1.1. Analisis Kompleksitas Algoritma Uniform Cost Search (UCS).....	29
4.1.2. Analisis Kompleksitas Algoritma Greedy Best First Search.....	29
4.1.3. Analisis Kompleksitas Algoritma A*.....	29
4.2. Hasil dan Analisis Percobaan.....	30
4.2.1. Konfigurasi Puzzle Sama, Algoritma Berbeda.....	30
4.2.2. Konfigurasi Puzzle Berbeda, Algoritma Uniform Cost Search.....	35
4.2.3. Konfigurasi Puzzle Berbeda, Algoritma Greedy Best First Search.....	38
4.2.4. Konfigurasi Puzzle Berbeda, Algoritma A*.....	43
BAB V.....	47
5.1. Heuristic Alternatif.....	47
BAB VI.....	48
LAMPIRAN.....	49
REFERENSI.....	50

BAB I

DESKRIPSI MASALAH

1.1 Algoritma *Pathfinding*

Algoritma *pathfinding* adalah algoritma yang digunakan dalam proses eksplorasi graf untuk menentukan jalur atau *path* paling optimal dari suatu titik tertentu menuju titik atau tujuan tertentu (atau dalam konteks tugas kecil ini adalah dari *start* hingga *finish* permainan) dengan mempertimbangkan *cost*, *constraints*, *obstacles*, dan lain-lain. Algoritma *pathfinding* atau *searching* dapat dibagi menjadi *Uninformed Search* dan *Informed Search*. *Uninformed Search* seringkali disebut *Blind Search* yang artinya teknik pencarian ini tidak memiliki informasi tambahan mengenai kondisi di luar dari yang telah disediakan oleh definisi persoalan. Algoritma ini melakukan *generate* dari *successor* dan membedakan *goal state* dan yang tidak. *Informed Search* seringkali disebut *Heuristic Search* yang artinya teknik pencarian ini menggunakan informasi spesifik kepada persoalan yang ada di samping dari definisi persoalan itu sendiri.

1.1.1. Algoritma *Uniform Cost Search* (UCS)

Algoritma *Uniform Cost Search* (UCS) adalah salah satu algoritma *Uninformed Search* yang mirip seperti *Breadth First Search*. Jika pada BFS tidak memperhitungkan *cost* antar titik pada graf, maka UCS memperhitungkan *cost* antar titik pada graf. Jika seluruh *edges* tidak memiliki *cost* yang sama, maka dapat digunakan UCS untuk menemukan solusi yang optimal yakni jalur dengan *cost* terkecil. Pada UCS, didefinisikan $g(n)$ sebagai fungsi yang menyatakan *cost* jalur dari *root* sampai *node* ke-n. Ekspansi *node* pada UCS dilakukan berdasarkan *cost* dari *root*. Hal ini dapat dilakukan dengan membuat *priority queue* berdasarkan *cost* jalurnya yakni $g(n)$.

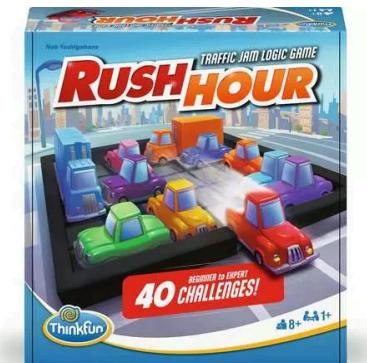
1.1.2. Algoritma *Greedy Best First Search*

Algoritma *Greedy Best First Search* adalah salah satu algoritma *Informed Search*. Pada *Informed Search*, didefinisikan $h(n)$ yang menyatakan estimasi *cost* dari *node* ke-n menuju *goal*. Jika n adalah *goal*, maka $h(n)$ bernilai nol. Ekspansi *node* pada GBFS dilakukan berdasarkan jarak terdekat dengan *goal* dan hanya dengan melihat kepada nilai dari fungsi heuristiknya.

1.1.3. Algoritma A*

Algoritma A* adalah salah satu algoritma *Informed Search*. Algoritma ini mengkombinasikan kebaikan dari algoritma UCS dan algoritma GBFS. Pada algoritma A*, didefinisikan $f(n) = g(n) + h(n)$. Ekspansi *node* pada A* melihat kombinasi *cost* dari *root* dan estimasi *cost* menuju *goal*.

1.2 Permainan Puzzle Rush Hour



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.

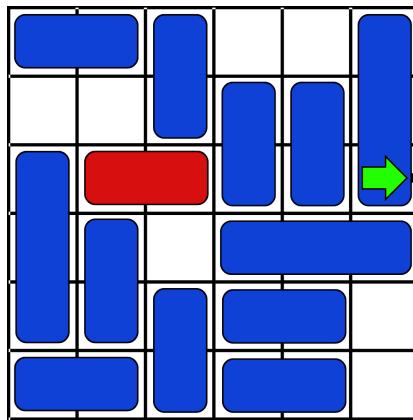
Komponen penting dari permainan Rush Hour terdiri dari:

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.

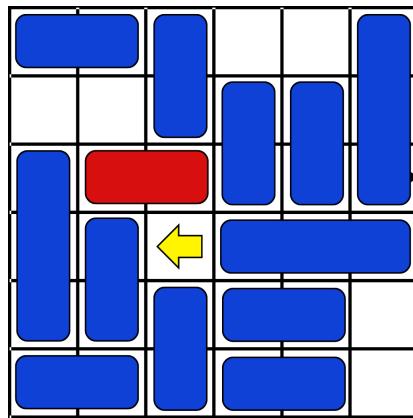
Ilustrasi kasus:

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.



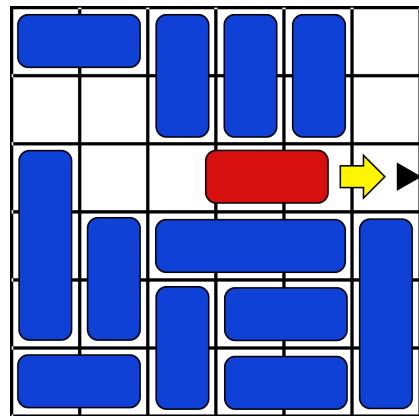
Gambar 2. Awal Permainan Game Rush Hour

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

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



Gambar 4. Pemain Menyelesaikan Permainan

BAB II

PENYELESAIAN

2.1. Langkah Penyelesaian Permainan Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Berikut adalah langkah-langkah penyelesaian puzzle Rush Hour menggunakan masing-masing algoritma *pathfinding*:

2.1.1. Penyelesaian Menggunakan Algoritma *Uniform Cost Search* (UCS)

Langkah-langkah penyelesaian puzzle Rush Hour menggunakan algoritma *Uniform Cost Search* (UCS):

1. Setelah membaca file konfigurasi puzzle, akan dilakukan inisialisasi *priority queue* dengan awalnya hanya berisi konfigurasi awal dari puzzle dan jumlah *node* yang diperiksa adalah nol. *Node* dalam algoritma ini adalah *state* dari papan puzzle.
2. Algoritma akan mengambil *state* dari *priority queue* yang memiliki banyak gerakan atau $g(n)$ terkecil, yakni mengambil *state* pada posisi pertama *priority queue*. Dari *state* tersebut, akan dihasilkan semua kemungkinan *state* selanjutnya atau semua gerakan yang mungkin dilakukan.
3. Jika *state* yang diperiksa belum pernah dikunjungi, maka akan ditambahkan ke dalam set *visited* agar tidak diperiksa kembali pada iterasi berikutnya.
4. Proses pencarian akan terus dilakukan dengan memeriksa apakah *state* yang diperiksa merupakan *goal*. Jika tidak, maka akan dilakukan langkah kedua hingga queue kosong atau jika *state* adalah *goal*, maka pencarian akan dihentikan dan ditemukan solusi yakni jalur dengan *cost* terkecil.

2.1.2. Penyelesaian Menggunakan Algoritma *Greedy Best First Search*

Langkah-langkah penyelesaian puzzle Rush Hour menggunakan algoritma *Greedy Best First Search* (GBFS):

1. Setelah membaca file konfigurasi puzzle, akan dilakukan inisialisasi *priority queue* dengan awalnya hanya berisi konfigurasi awal dari puzzle dan jumlah *node* yang diperiksa adalah nol. *Node* dalam algoritma ini adalah *state* dari papan puzzle.
2. Algoritma akan mengambil *state* dari *priority queue* yang memiliki estimasi jarak *primary piece* ke *exit* atau $h(n)$ terkecil, yakni mengambil *state* pada posisi pertama *priority queue*. Fungsi $h(n)$ ditentukan berdasarkan heuristik yang digunakan. Dari *state* tersebut, akan dihasilkan semua kemungkinan *state*

selanjutnya atau semua gerakan yang mungkin dilakukan.

3. Jika *state* yang diperiksa belum pernah dikunjungi, maka akan ditambahkan ke dalam set *visited* agar tidak diperiksa kembali pada iterasi berikutnya.
4. Proses pencarian akan terus dilakukan dengan memeriksa apakah *state* yang diperiksa merupakan *goal*. Jika tidak, maka akan dilakukan langkah kedua hingga queue kosong atau jika *state* adalah *goal*, maka pencarian akan dihentikan dan ditemukan solusi.

2.1.3. Penyelesaian Menggunakan Algoritma A*

Langkah-langkah penyelesaian puzzle Rush Hour menggunakan algoritma *Greedy Best First Search* (GBFS):

1. Setelah membaca file konfigurasi puzzle, akan dilakukan inisialisasi *priority queue* dengan awalnya hanya berisi konfigurasi awal dari puzzle dan jumlah *node* yang diperiksa adalah nol. *Node* dalam algoritma ini adalah *state* dari papan puzzle.
2. Algoritma akan mengambil *state* dari *priority queue* yang memiliki *total cost of path through n to goal* atau $f(n) = g(n) + h(n)$ terkecil, yakni mengambil *state* pada posisi pertama *priority queue*. Dari *state* tersebut, akan dihasilkan semua kemungkinan *state* selanjutnya atau semua gerakan yang mungkin dilakukan.
3. Jika *state* yang diperiksa belum pernah dikunjungi, maka akan ditambahkan ke dalam set *visited* agar tidak diperiksa kembali pada iterasi berikutnya.
4. Proses pencarian akan terus dilakukan dengan memeriksa apakah *state* yang diperiksa merupakan *goal*. Jika tidak, maka akan dilakukan langkah kedua hingga queue kosong atau jika *state* adalah *goal*, maka pencarian akan dihentikan dan ditemukan solusi.

2.2. Analisis Algoritma Pathfinding

Pada penyelesaian puzzle Rush Hour menggunakan algoritma *pathfinding*, $g(n)$ didefinisikan sebagai banyak gerakan dari *initial state* sampai *state* ke-n, $h(n)$ didefinisikan sebagai fungsi heuristik yang terdiri atas Manhattan Distance dan Blocking Vehicles, sehingga $f(n) = g(n) + h(n)$ atau menggabungkan banyak gerakan yang dilakukan dan estimasi jarak dari *primary piece* ke *exit*.

Menurut salindia kuliah, algoritma A* *admissible* atau selalu menemukan solusi optimal jika $h(n) \leq h^*(n)$ atau estimasi biaya $h(n)$ ke *goal* tidak pernah melebihi biaya sesungguhnya $h^*(n)$ ke *goal*. Heuristik yang digunakan algoritma A* pada tugas kecil ini yakni Manhattan Distance dan Blocking Vehicles. Heuristik Manhattan Distance tidak memperhitungkan mobil/*piece* lain yang menghalangi *primary piece* tetapi hanya jarak *primary piece* ke *exit*. Misalnya $h^*(n)$ dari *state* n adalah minimum 5 gerakan (3 gerakan untuk memindahkan mobil

penghalang dan 2 gerakan untuk menggerakan *primary piece* ke *exit*) dan $h(n)$ dari state n adalah hanya 2 gerakan. Karena $h(n) \leq h^*(n)$, maka algoritma A* yang digunakan *admissible*. Sementara itu, heuristik Blocking Vehicles tidak memperhitungkan jarak *primary piece* ke *exit* tetapi hanya jumlah mobil/*piece* yang menghalangi *primary piece* ke *exit*. Misalnya $h^*(n)$ dari state n adalah minimum 7 gerakan (3 gerakan untuk memindahkan mobil penghalang dan 4 gerakan untuk menggerakan *primary piece* ke *exit*) dan $h(n)$ dari state n adalah hanya 3 gerakan. Karena $h(n) \leq h^*(n)$, maka algoritma A* yang digunakan *admissible*.

Pada penyelesaian puzzle Rush Hour, *node* yang dibangkitkan dan *path* yang dihasilkan dari algoritma UCS bisa saja berbeda dengan algoritma BFS. Meskipun *cost* jalur kedua *state* berbeda bernilai sama karena satu gerakan dianggap *cost*-nya satu, *state* yang diperiksa bergantung saat penambahan *state* ke *priority queue*.

Secara teoritis, algoritma A* lebih efisien dibandingkan algoritma UCS. Hal ini karena algoritma UCS melakukan *Blind Search* atau tidak ada informasi tambahan sehingga menyusuri semua kemungkinan jalur dengan *cost* terkecil. Sementara algoritma A* menggunakan gabungan jalur dengan *cost* terkecil dan heuristik, heuristik ini menjadi informasi tambahan yang dapat memandu pencarian ke arah *goal* dengan lebih cepat. Heuristik pada algoritma A* memungkinkan eksplorasi *node* yang lebih sedikit.

Secara teoritis, algoritma *Greedy Best First Search* tidak dapat menjamin solusi optimal untuk penyelesaian puzzle Rush Hour. Hal ini karena algoritma GBFS hanya menggunakan nilai $h(n)$ dan berfokus mendekati *goal* secepat mungkin. Konsekuensinya, algoritma GBFS dapat terjebak dalam jalur yang tampak memiliki heuristik dengan nilai kecil tapi sebenarnya memerlukan banyak langkah sehingga solusi yang ditemukan bisa saja lebih panjang langkahnya daripada solusi optimal.

BAB III

IMPLEMENTASI

3.1. Spesifikasi Teknis Program

3.1.1. Struktur *Repository*

```
└── Tucil3_13523060_13523070
    ├── bin
    │   ├── Algorithm.class
    │   ├── Main.class
    │   ├── Piece.class
    │   ├── Reader.class
    │   └── State.class
    ├── doc
    │   └── Tucil3_13523060_13523070.pdf
    └── src
        ├── Algorithm.java
        ├── Main.java
        ├── Piece.java
        ├── Reader.java
        └── State.java
    └── test
        └── solution
            ├── solusi1_astar_heuristic1.txt
            ├── solusi1_astar_heuristic2.txt
            ├── solusi1_greedy_heuristic1.txt
            ├── solusi1_greedy_heuristic2.txt
            ├── solusi1_ucs.txt
            ├── solusi2_ucs.txt
            ├── solusi3_ucs.txt
            ├── solusi4_greedy_heuristic1.txt
            ├── solusi4_greedy_heuristic2.txt
            ├── solusi4_ucs.txt
            ├── solusi5_greedy_heuristic1.txt
            ├── solusi5_greedy_heuristic2.txt
            ├── solusi5_ucs.txt
            └── solusi6_astar_heuristic1.txt
```

```

|__ solusi6_astar_heuristic2.txt
|__ solusi7_astar_heuristic1.txt
|__ solusi7_astar_heuristic2.txt
|__ test1.txt
|__ test2.txt
|__ test3.txt
|__ test4.txt
|__ test5.txt
|__ test6.txt
|__ test7.txt
|__ test8.txt
|__ README.md

```

3.1.2. Source Code

1. Reader.java

```

1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.util.ArrayList;
4 import java.util.HashSet;
5 import java.util.List;
6
7 public class Reader {
8     public static int rows, cols, pieceCount;
9     private static List<Piece> extractedPieces = new ArrayList<>();
10    private static char[][] board;
11    private static boolean hasLeftExit = false;
12    private static boolean hasRightExit = false;
13    private static boolean hasTopExit = false;
14    private static boolean hasBottomExit = false;
15    private static int exitRow = -1;
16    private static int exitCol = -1;
17    public static boolean readInputFile(String filePath) {
18        // System.out.println("DEBUG: Starting to read file: " + filePath);
19        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
20            String[] dimensions = br.readLine().split(regex:" ");
21            rows = Integer.parseInt(dimensions[0]);
22            cols = Integer.parseInt(dimensions[1]);
23            // System.out.println("Rows: " + rows + ", Cols: " + cols);
24            pieceCount = Integer.parseInt(br.readLine());
25            pieceCount += 1; // + PRIMARY PIECE
26            // System.out.println("DEBUG: Total expected pieces: " + pieceCount);
27
28            ArrayList<String> boardLines = new ArrayList<>();
29            String line;
30            while ((line = br.readLine()) != null) {
31                if (line.isEmpty()) continue;
32                boardLines.add(line);
33                // System.out.println("DEBUG: Read board line: " + line);
34            }
35            // System.out.println("DEBUG: Total board lines read: " + boardLines.size());
36

```

```

37     findExitLocations(boardLines);
38
39     if (!hasTopExit) {
40         boardLines.remove(index:0); // skip first line
41     }
42
43     board = new char[rows][cols];
44     for (int i = 0; i < rows; i++) {
45         char[] rowChars = boardLines.get(i).toCharArray();
46         for (int j = 0; j < cols; j++) {
47             if (hasLeftExit && j < rowChars.length - 1) {
48                 board[i][j] = rowChars[j + 1];
49             } else if (!hasLeftExit && j < rowChars.length) {
50                 board[i][j] = rowChars[j];
51             } else {
52                 board[i][j] = '.';
53             }
54         }
55     }
56
57     // System.out.println("DEBUG: Board:");
58     for (int i = 0; i < rows; i++) {
59         // System.out.println("DEBUG: " + new String(board[i]));
60     }
61
62     // Extract unique pieces
63     HashSet<Character> uniquePieces = new HashSet<>();
64     for (int i = 0; i < rows; i++) {
65         for (int j = 0; j < cols; j++) {
66             if (board[i][j] != '.' && board[i][j] != 'K' && board[i][j] != ' ') {
67                 uniquePieces.add(board[i][j]);
68             }
69         }
70     }
71
72
73     List<Piece> pieces = new ArrayList<>();
74     for (char pieceChar : uniquePieces) {
75         // System.out.println("DEBUG: Extracting piece: " + pieceChar);
76         Piece extractedPiece = extractPiece(pieceChar);
77         if (extractedPiece != null) {
78             pieces.add(extractedPiece);
79             // System.out.println("DEBUG: Added piece " + pieceChar);
80         } else {
81             // System.out.println("DEBUG: Failed to extract piece " + pieceChar);
82         }
83     }
84
85     extractedPieces = pieces;
86
87     // System.out.println("DEBUG: Total pieces: " + pieces.size() + "/" + pieceCount);
88     return pieces.size() == pieceCount;
89 } catch (Exception e) {
90     System.out.println("Exception occurred: " + e.getMessage());
91     // e.printStackTrace(); // Error Debug
92     return false;
93 }
94
95 }
96
97
98     private static Piece extractPiece(char pieceChar) {
99         int minRow = rows, maxRow = -1, minCol = cols, maxCol = -1;
100
101        for (int i = 0; i < rows; i++) {
102            for (int j = 0; j < cols; j++) {
103                if (board[i][j] == pieceChar) {
104                    minRow = Math.min(minRow, i);
105                    maxRow = Math.max(maxRow, i);
106                    minCol = Math.min(minCol, j);
107                    maxCol = Math.max(maxCol, j);
108                }
109            }
110
111        int height = maxRow - minRow + 1;
112        int width = maxCol - minCol + 1;
113
114        boolean isHorizontal = width > height;
115        int length = isHorizontal ? width : height;
116
117        return new Piece(pieceChar, minRow, minCol, length, isHorizontal);
118    }

```

```

120     public static State getInitialState() {
121         if (extractedPieces == null || extractedPieces.isEmpty()) {
122             System.out.println(x:"Initial State gagal dibuat");
123             return null;
124         }
125
126         // Sort Primary piece at 0
127         extractedPieces.sort((a, b) -> {
128             if (a.getId() == 'P') return -1;
129             if (b.getId() == 'P') return 1;
130             return Character.compare(a.getId(), b.getId());
131         });
132
133         return new State(extractedPieces, cost:0, heuristic:0, parent:null, cols, rows);
134     }
135
136     private static void findExitLocations(ArrayList<String> boardLines) {
137         for (int i = 0; i < boardLines.size(); i++) {
138             String line = boardLines.get(i);
139             for (int j = 0; j < line.length(); j++) {
140                 if (line.charAt(j) == 'K') {
141                     if (i == 0) { // Top
142                         hasTopExit = true;
143                         exitRow = 0;
144                         exitCol = j;
145                         // System.out.println("DEBUG: Exit di atas: " + exitCol);
146                     } else if (i == boardLines.size() - 1) { // Bottom
147                         hasBottomExit = true;
148                         exitRow = rows - 1;
149                         exitCol = j;
150                         // System.out.println("DEBUG: Exit di bawah: " + exitCol);
151                     } else if (j == 0) { // Left
152                         hasLeftExit = true;
153                         exitRow = i;
154                         exitCol = 0;
155                         // System.out.println("DEBUG: Exit di kiri: " + exitRow);
156                     } else if (j == line.length() - 1) { // Right
157                         hasRightExit = true;
158                         exitRow = i;
159                         exitCol = cols - 1;
160                         // System.out.println("DEBUG: Exit di kanan: " + exitRow);
161                     }
162                 }
163             }
164         }
165
166         if (!hasLeftExit && !hasRightExit && !hasTopExit && !hasBottomExit) {
167             System.out.println(x:"Exit tidak ditemukan, program akan keluar.");
168             System.exit(status:0);
169         } else {
170             // System.out.println("DEBUG: Exit [" + exitRow + "," + exitCol + "]");
171         }
172     }
173
174     public static boolean hasLeftExit() { return hasLeftExit; }
175     public static boolean hasRightExit() { return hasRightExit; }
176     public static boolean hasTopExit() { return hasTopExit; }
177     public static boolean hasBottomExit() { return hasBottomExit; }
178     public static int getExitRow() { return exitRow; }
179     public static int getExitCol() { return exitCol; }
180 }

```

2. Piece.java

```

1  public class Piece {
2      private final char id;
3      private final int row;
4      private final int col;
5      private final int length;
6      private final boolean isHorizontal;
7
8      public Piece(char id, int row, int col, int length, boolean isHorizontal) {
9          this.id = id;
10         this.row = row;
11         this.col = col;
12         this.length = length;
13         this.isHorizontal = isHorizontal;
14     }
15
16     public char getId() { return id; }
17     public int getRow() { return row; }
18     public int getCol() { return col; }
19     public int getLength() { return length; }
20     public boolean isHorizontal() { return isHorizontal; }
21
22     @Override
23     public String toString() {
24         return id + " at (" + row + ", " + col + ") " + (isHorizontal ? "H" : "V") + " len=" + length;
25     }
26
27     public Piece move(int direction) {
28         return isHorizontal
29             ? new Piece(id, row, col + direction, length, isHorizontal:true)
30             : new Piece(id, row + direction, col, length, isHorizontal:false);
31     }
32
33     public Piece copy() {
34         return new Piece(id, row, col, length, isHorizontal);
35     }
36 }

```

3. State.java

```

1  import java.util.ArrayList;
2  import java.util.Arrays;
3  import java.util.Comparator;
4  import java.util.List;
5  import java.util.Objects;
6
7  public class State implements Comparable<State> {
8      private final List<Piece> pieces;
9      private final int cost;           // g(n): path cost
10     private final int heuristic;    // h(n): estimated cost to goal
11     private final State parent;     // path reconstruction
12     private int cols;
13     private int rows;
14
15     public State(List<Piece> pieces, int cost, int heuristic, State parent, int cols, int rows) {
16         this.pieces = new ArrayList<>(pieces);
17         for (Piece p : pieces) {
18             this.pieces.add(new Piece(p.getId(), p.getRow(), p.getCol(), p.getLength(), p.isHorizontal()));
19         }
20         this.cost = cost;
21         this.heuristic = heuristic;
22         this.parent = parent;
23         this.cols = cols;
24         this.rows = rows;
25     }
26
27     public List<Piece> getPieces() { return pieces; }
28     public int getCost() { return cost; }
29     public int getHeuristic() { return heuristic; }
30     public int getTotalCost() { return cost + heuristic; }
31     public State getParent() { return parent; }
32     public int getCols() { return cols; }
33     public int getRows() { return rows; }
34
35     // Mengecek apakah state sekarang adalah goal state
36     public boolean isGoal() {
37         Piece primary = pieces.get(index:0);
38
39         if (Reader.hasRightExit()) {
40             int endCol = primary.getCol() + primary.getLength() - 1;
41             return endCol == cols - 1 && primary.getRow() == Reader.getExitRow();
42         }
43         else if (Reader.hasLeftExit()) {
44             return primary.getCol() == 0 && primary.getRow() == Reader.getExitRow();
45         }
46         else if (Reader.hasTopExit()) {
47             return primary.getRow() == 0 && primary.getCol() == Reader.getExitCol();
48         }
49         else if (Reader.hasBottomExit()) {
50             int endRow = primary.getRow() + (primary.isHorizontal() ? 0 : primary.getLength() - 1);
51             return endRow == this.rows - 1 && primary.getCol() == Reader.getExitCol();
52         } else {
53             return false;
54         }
55     }

```

```

7   public class State implements Comparable<State> {
37     public boolean isGoal() {
55       }
56     }
57
58     @Override
59     public int compareTo(State other) {
60       return Integer.compare(this.getTotalCost(), other.getTotalCost());
61     }
62
63     @Override
64     public boolean equals(Object o) {
65       if (!(o instanceof State)) return false;
66       State other = (State) o;
67       return this.getPieceString().equals(other.getPieceString());
68     }
69
70     @Override
71     public int hashCode() {
72       return Objects.hash(getPieceString());
73     }
74
75     private String getPieceString() {
76       StringBuilder sb = new StringBuilder();
77       for (Piece p : pieces.stream().sorted(Comparator.comparing(Piece::getId)).toList()) {
78         sb.append(p.getId()).append(p.getRow()).append(p.getCol());
79       }
80       return sb.toString();
81     }
82
83     public void printState(int rows, int cols) {
84       char[][] grid = new char[rows][cols];
85       for (char[] row : grid) Arrays.fill(row, val:'.');
86       for (Piece p : pieces) {
87         int r = p.getRow(), c = p.getCol();
88         for (int i = 0; i < p.getLength(); i++) {
89           if (p.isHorizontal()) grid[r][c + i] = p.getId();
90           else grid[r + i][c] = p.getId();
91         }
92       }
93
94       for (char[] row : grid) {
95         System.out.println(new String(row));
96       }
97       System.out.println();
98     }
99
100    // Menghasilkan semua state yang mungkin dari state sekarang
101    public List<State> generateNextStates(int rows, int cols) {
102      List<State> nextStates = new ArrayList<>();
103
104      for (int i = 0; i < pieces.size(); i++) {
105        Piece p = pieces.get(i);
106
107        // Gerakan
108        for (int dir = -1; dir <= 1; dir += 2) {
109          int newRow = p.getRow();
110          int newCol = p.getCol();
111
112          boolean canMove = true;
113
114          if (p.isHorizontal()) {
115            newCol += dir;
116            for (int j = 0; j < p.getLength(); j++) {
117              int colToCheck = (dir == -1) ? newRow : newCol + p.getLength() - 1;
118              if (colToCheck < 0 || colToCheck >= cols || isOccupied(p, newRow, colToCheck)) {
119                canMove = false;
120                break;
121              }
122            }
123          } else {
124            newRow += dir;
125            for (int j = 0; j < p.getLength(); j++) {
126              int rowToCheck = (dir == -1) ? newRow : newRow + p.getLength() - 1;
127              if (rowToCheck < 0 || rowToCheck >= rows || isOccupied(p, rowToCheck, newCol)) {
128                canMove = false;
129                break;
130              }
131            }
132          }
133
134          if (canMove) {
135            List<Piece> newPieces = new ArrayList<>();
136            for (int k = 0; k < pieces.size(); k++) {
137              if (k == i) {
138                // Move the piece
139                newPieces.add(p.move(dir));
140              } else {
141                newPieces.add(pieces.get(k).copy());
142              }
143            }
144
145            State next = new State(newPieces, this.cost + 1, heuristic:0, this, this.cols, this.rows);
146            nextStates.add(next);
147          }
148        }
149      }
150
151      return nextStates;
152    }

```

```

154     public class State implements Comparable<State> {
155         private boolean isOccupied(Piece movingPiece, int row, int col) {
156             for (Piece other : pieces) {
157                 if (other == movingPiece) continue;
158
159                 int r = other.getRow();
160                 int c = other.getCol();
161                 for (int i = 0; i < other.getLength(); i++) {
162                     int rr = r + i;
163                     if (other.isHorizontal()) cc += i;
164                     else rr += i;
165
166                     if (rr == row && cc == col) return true;
167                 }
168             }
169             return false;
170         }
171
172         @Override
173         public String toString() {
174             StringBuilder sb = new StringBuilder();
175             for (Piece p : pieces) {
176                 sb.append(p.getId()).append(p.getRow()).append(p.getCol()).append(str);
177             }
178             return sb.toString();
179         }
180
181         public char getMovedPiece() {
182             if (parent != null) {
183                 for (int i = 0; i < pieces.size(); i++) {
184                     Piece currentPiece = pieces.get(i);
185                     Piece parentPiece = parent.getPieces().get(i);
186
187                     if (currentPiece.getRow() != parentPiece.getRow() ||
188                         currentPiece.getCol() != parentPiece.getCol()) {
189                         return currentPiece.getId();
190                     }
191                 }
192             }
193             return ' ';
194         }
195
196         public String getMovedPieceDirection() {
197             String moveDirection = "";
198             if (parent != null) {
199                 for (int i = 0; i < pieces.size(); i++) {
200                     Piece currentPiece = pieces.get(i);
201                     Piece parentPiece = parent.getPieces().get(i);
202
203                     if (currentPiece.getRow() == parentPiece.getRow() ||
204                         currentPiece.getCol() == parentPiece.getCol()) {
205                         // Menentukan arah gerakan piece
206                         if (currentPiece.isHorizontal()) {
207                             if (currentPiece.getCol() > parentPiece.getCol()) {
208                                 moveDirection = "Kanan";
209                             } else {
210                                 moveDirection = "Kiri";
211                             }
212                         } else {
213                             if (currentPiece.getRow() > parentPiece.getRow()) {
214                                 moveDirection = "Bawah";
215                             } else {
216                                 moveDirection = "Atas";
217                             }
218                         }
219                     }
220                 }
221             }
222             return moveDirection;
223         }
224
225         public String getBoard(int rows, int cols) {
226             char[][] grid = new char[rows][cols];
227             for (char[] row : grid) Arrays.fill(row, val);
228             char[][] parentGrid = new char[rows][cols];
229             for (char[] row : parentGrid) Arrays.fill(row, val);
230
231             char movedPiece = getMovedPiece();
232             final String BG_YELLOW = "\u001B[43m";
233
234             // Membandingkan titik / sel kosong
235             for (Piece p : pieces) {
236                 int r = p.getRow(), c = p.getCol();
237                 for (int i = 0; i < p.getLength(); i++) {
238                     if (p.isHorizontal()) grid[r][c + i] = p.getId();
239                     else grid[r + i][c] = p.getId();
240                 }
241             }
242             int x = -1, y = -1;
243             if (parent != null) {
244                 for (Piece p : parent.getPieces()) {
245                     int r = p.getRow(), c = p.getCol();
246                     for (int i = 0; i < p.getLength(); i++) {
247                         if (p.isHorizontal()) parentGrid[r][c + i] = p.getId();
248                         else parentGrid[r + i][c] = p.getId();
249                     }
250                 }
251             }
252             for (int i = 0; i < rows; i++) {
253                 for (int j = 0; j < cols; j++) {
254                     if (grid[i][j] == ' ' && parentGrid[i][j] != ' ') {
255                         grid[i][j] = movedPiece;
256                     }
257                 }
258             }
259             return new String(new CharArrayWrapper(grid));
260         }
261     }

```

```

7   public class State implements Comparable<State> {
224     public String getBoard(int rows, int cols) {
250       for (int i = 0; i < rows; i++) {
251         for (int j = 0; j < cols; j++) {
252           if (grid[i][j] == '.' && parentGrid[i][j] != '.') {
253             x = i;
254             y = j;
255           }
256         }
257       }
258     }
259   }
260
261   int exitlocation = 0;
262   if (Reader.hasTopExit() || Reader.hasBottomExit()) {
263     exitlocation = Reader.getExitCol();
264   } else if (Reader.hasLeftExit() || Reader.hasRightExit()) {
265     exitlocation = Reader.getExitRow();
266   }
267
268   StringBuilder sb = new StringBuilder();
269   if (Reader.hasBottomExit()) {
270     // Print board
271     for (int r = 0; r < rows; r++) {
272       for (int c = 0; c < cols; c++) {
273         char cell = grid[r][c];
274         if (cell == movedPiece) { // highlight piece yang gerak
275           sb.append(BG_YELLOW).append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
276         } else if (cell == '.' && r == x && c == y) {
277           sb.append(BG_YELLOW).append(cell).append(WARNA_DEFAULT);
278         } else {
279           sb.append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
280         }
281       }
282       sb.append(str:"\n");
283     }
284
285     // Print bottom exit
286     for (int i = 0; i < cols; i++) {
287       if (i == exitlocation) {
288         sb.append(getWarnaPiece(pieceID:'K')).append(str:"K").append(WARNA_DEFAULT);
289       } else {
290         sb.append(str:" ");
291       }
292     }
293     sb.append(str:"\n");
294   } else if (Reader.hasTopExit()) {
295     // Print top exit
296     for (int i = 0; i < cols; i++) {
297       if (i == exitlocation) {
298         sb.append(getWarnaPiece(pieceID:'K')).append(str:"K").append(WARNA_DEFAULT);
299       } else {
300         sb.append(str:" ");
301       }
302     }
303     sb.append(str:"\n");
304   }
305
306   // Print board
307   for (int r = 0; r < rows; r++) {
308     for (int c = 0; c < cols; c++) {
309       char cell = grid[r][c];
310       if (cell == movedPiece) { // highlight piece yang gerak
311         sb.append(BG_YELLOW).append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
312       } else if (cell == '.' && r == x && c == y) {
313         sb.append(BG_YELLOW).append(cell).append(WARNA_DEFAULT);
314       } else {
315         sb.append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
316       }
317     }
318     sb.append(str:"\n");
319   }
320
321   else if (Reader.hasLeftExit()) {
322     // Print board with left exit
323     for (int r = 0; r < rows; r++) {
324       if (r == exitlocation) {
325         sb.append(getWarnaPiece(pieceID:'K')).append(str:"K").append(WARNA_DEFAULT);
326       } else {
327         sb.append(str:" ");
328       }
329     }
330
331     for (int c = 0; c < cols; c++) {
332       char cell = grid[r][c];
333       if (cell == movedPiece) { // highlight piece yang gerak
334         sb.append(BG_YELLOW).append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
335       } else if (cell == '.' && r == x && c == y) {
336         sb.append(BG_YELLOW).append(cell).append(WARNA_DEFAULT);
337       } else {
338         sb.append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
339       }
340     }
341     sb.append(str:"\n");
342   }
343   else if (Reader.hasRightExit()) {
344     // Print board with right exit
345     for (int r = 0; r < rows; r++) {

```

```

7   public class State implements Comparable<State> {
224     public String getBoard(int rows, int cols) {
343       else if (Reader.hasRightExit()) {
344         // Print board with right exit
345         for (int r = 0; r < rows; r++) {
346           for (int c = 0; c < cols; c++) {
347             char cell = grid[r][c];
348             if (cell == movedPiece) { // highlight piece yang gerak
349               sb.append(B6_YELLOW).append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
350             } else if (cell == '.' && r == x && c == y) {
351               sb.append(B6_YELLOW).append(cell).append(WARNA_DEFAULT);
352             } else {
353               sb.append(getWarnaPiece(cell)).append(cell).append(WARNA_DEFAULT);
354             }
355           }
356           if (r == exitLocation) {
357             sb.append(getWarnaPiece(pieceID:'K')).append(str:"K").append(WARNA_DEFAULT);
358           }
359           sb.append(str:"\n");
360         }
361       }
362     }
363     return sb.toString();
364   }
365 }
366
367 public static String getWarnaPiece(char pieceID) {
368   if (pieceID == '.'){
369     return WARNA_DEFAULT;
370   }
371   if (pieceID < 'A' || pieceID > 'Z') {
372     return WARNA_DEFAULT;
373   } else {
374     int indeks = pieceID - 'A';
375     return WARNA[indeks];
376   }
377 }
378
379 public static final String[] WARNA = {
380   "\u0001B[38;2;255;255;25m", // Putih, A
381   "\u0001B[38;2;166;166;16m", // Abu-abu, B
382   "\u0001B[38;2;255;145;76m", // Oranye, C
383   "\u0001B[38;2;255;155;155m", // Kuning, D
384   "\u0001B[38;2;161;208;25m", // Hijau, E
385   "\u0001B[38;2;151;159;124m", // Biru Muda, F
386   "\u0001B[38;2;174;173m", // Biru Tua, G
387   "\u0001B[38;2;255;101;195m", // Pink, H
388   "\u0001B[38;2;148;82;255m", // Ungu, I
389   "\u0001B[38;2;255;87;87m", // Terracotta, J
390   "\u0001B[38;2;126;217;86m", // Hijau Muda, K (EXIT)
391   "\u0001B[38;2;148;255;162m", // Kuning Pucat, L
392   "\u0001B[38;2;254;189;89m", // Kuning Tua, M
393 }
394
395 public class State implements Comparable<State> {
396   public static final String[] WARNA = {
397     "\u0001B[38;2;148;73;10m", // Coklat, N
398     "\u0001B[38;2;102;255;114m", // Lilac, O
399     "\u0001B[38;2;255;49;49m", // Merah, P (PRIMARY)
400     "\u0001B[38;2;137;42m", // Hijau Tua, Q
401     "\u0001B[38;2;92;25;230m", // Cyan, R
402     "\u0001B[38;2;151;178m", // Turquoise, S
403     "\u0001B[38;2;56;182;255m", // Biru Langit, T
404     "\u0001B[38;2;82;113;255m", // Indigo, U
405     "\u0001B[38;2;245;57;255m", // Magenta, V
406     "\u0001B[38;2;240;175;255m", // Lavender, W
407     "\u0001B[38;2;203;107;230m", // Violet, X
408     "\u0001B[38;2;93;23;235m", // Ungu Tua, Y
409     "\u0001B[38;2;128;8;0m" // Maroon, Z
410   };
411 }
412
413 public static final String WARNA_DEFAULT = "\u0001B[0m";
414
415 public String getBoardPlain(int rows, int cols) {
416   char[][] grid = new char[rows][cols];
417   for (char[] row : grid) Arrays.fill(row, val:'.');
418   char[][] parentGrid = new char[rows][cols];
419   for (char[] row : parentGrid) Arrays.fill(row, val:'.');
420
421   char movedPiece = getMovedPiece();
422
423   for (Piece p : pieces) {
424     int r = p.getRow(), c = p.getCol();
425     for (int i = 0; i < p.getLength(); i++) {
426       if (p.isHorizontal()) parentGrid[r][c + i] = p.getId();
427       else grid[r + i][c] = p.getId();
428     }
429   }
430
431   int x = -1, y = -1;
432   if (parent != null) {
433     for (Piece p : parent.pieces) {
434       int r = p.getRow(), c = p.getCol();
435       for (int i = 0; i < p.getLength(); i++) {
436         if (p.isHorizontal()) parentGrid[r][c + i] = p.getId();
437         else parentGrid[r + i][c] = p.getId();
438       }
439     }
440
441     for (int i = 0; i < rows; i++) {
442       for (int j = 0; j < cols; j++) {
443         if (grid[i][j] == '.' && parentGrid[i][j] != '.') {
444           x = i;
445           y = j;
446         }
447       }
448     }
449   }
450 }

```

```

418     public String getBoardPlain(int rows, int cols) {
419
420         int exitLocation = 0;
421         if (Reader.hasTopExit() || Reader.hasBottomExit()) {
422             exitLocation = Reader.getExitCol();
423         } else if (Reader.hasLeftExit() || Reader.hasRightExit()) {
424             exitLocation = Reader.getExitRow();
425         }
426
427         StringBuilder sb = new StringBuilder();
428         if (Reader.hasBottomExit()) {
429             for (int r = 0; r < rows; r++) {
430                 for (int c = 0; c < cols; c++) {
431                     sb.append(grid[r][c]);
432                 }
433                 sb.append(str("\n"));
434             }
435             for (int i = 0; i < cols; i++) {
436                 sb.append(i == exitLocation ? "K" : " ");
437             }
438             sb.append(str("\n"));
439         } else if (Reader.hasTopExit()) {
440             for (int i = 0; i < cols; i++) {
441                 sb.append(i == exitLocation ? "K" : " ");
442             }
443             sb.append(str("\n"));
444             for (int r = 0; r < rows; r++) {
445                 for (int c = 0; c < cols; c++) {
446                     sb.append(grid[r][c]);
447                 }
448                 sb.append(str("\n"));
449             }
450         } else if (Reader.hasLeftExit()) {
451             for (int r = 0; r < rows; r++) {
452                 sb.append(r == exitLocation ? "K" : " ");
453                 for (int c = 0; c < cols; c++) {
454                     sb.append(grid[r][c]);
455                 }
456                 sb.append(str("\n"));
457             }
458         } else if (Reader.hasRightExit()) {
459             for (int r = 0; r < rows; r++) {
460                 for (int c = 0; c < cols; c++) {
461                     sb.append(grid[r][c]);
462                 }
463                 sb.append(r == exitLocation ? "K" : " ");
464             }
465             sb.append(str("\n"));
466         }
467         return sb.toString();
468     }
469 }

```

4. Algorithm.java

```

import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;

public class Algorithm {
    private static int nodesVisited = 0;

    public static int getNodesVisited() {
        return nodesVisited;
    }

    // Greedy Best First Search
    // hanya menggunakan h(n)
    public static State greedy(State initial, int rows, int cols, int heuristic) {
        PriorityQueue<State> queue;
        if (heuristic == 1) {
            queue = new PriorityQueue<>(Comparator.comparingInt(Algorithm::manhattanDistance));
        } else if (heuristic == 2) {
            queue = new PriorityQueue<>(Comparator.comparingInt(Algorithm::blockingVehicles));
        } else {
            throw new IllegalArgumentException("Invalid heuristic type");
        }
        Set<String> visited = new HashSet<>();

        queue.add(initial);
        nodesVisited = 0;

        while (!queue.isEmpty()) {
            State current = queue.poll();
            if (visited.contains(current.toString())) continue;
            visited.add(current.toString());
            nodesVisited++;

            if (current.isGoal()) {
                return current;
            }

            for (State next : current.generateNextStates(rows, cols)) {
                if (!visited.contains(next.toString())) {
                    queue.add(next);
                }
            }
        }
        return null;
    }
}

```

```
// Uniform Cost Search
// hanya menggunakan g(n)
public static State ucs(State initial, int rows, int cols) {
    PriorityQueue<State> frontier = new PriorityQueue<>(Comparator.comparingInt(State::getCost));
    Set<String> visited = new HashSet<>();

    frontier.add(initial); // inisialisasi dengan state awal
    nodesVisited = 0;

    while (!frontier.isEmpty()) {
        State current = frontier.poll();

        if (current.isGoal()) {
            return current;
        }

        if (visited.contains(current.toString())) continue;
        visited.add(current.toString());
        nodesVisited++;

        for (State neighbor : current.generateNextStates(rows, cols)) {
            if (!visited.contains(neighbor.toString())) {
                frontier.add(neighbor); // tambahkan tetangga ke queue
            }
        }
    }

    return null;
}
```

```

// A* Search
// menggunakan f(n) = g(n) + h(n)
public static State aStar(State initial, int rows, int cols, int heuristic) {
    PriorityQueue<State> queue;
    if (heuristic == 1) {
        // Use Manhattan Distance as the heuristic
        queue = new PriorityQueue<>(Comparator.comparingInt(state → state.getCost() + manhattanDistance(state)));
    } else if (heuristic == 2) {
        // Use Blocking Vehicles Heuristic as the heuristic
        queue = new PriorityQueue<>(Comparator.comparingInt(state → state.getCost() + blockingVehicles(state)));
    } else {
        throw new IllegalArgumentException("Invalid heuristic type");
    }
    Set<String> visited = new HashSet<>();
    queue.add(initial);
    nodesVisited = 0;

    while (!queue.isEmpty()) {
        State current = queue.poll();
        if (visited.contains(current.toString())) continue;
        visited.add(current.toString());
        nodesVisited++;

        if (current.isGoal()) {
            return current;
        }

        for (State next : current.generateNextStates(rows, cols)) {
            if (!visited.contains(next.toString())) {
                queue.add(next);
            }
        }
    }
    return null;
}

```

```
// Heuristic functions
private static int manhattanDistance(State state) {
    Piece primary = state.getPieces().get(0);

    // Right exit
    if (Reader.hasRightExit()) {
        int endCol = primary.getCol() + primary.getLength() - 1;
        int rowDistance = Math.abs(primary.getRow() - Reader.getExitRow());
        int colDistance = (state.getCols() - 1) - endCol;
        return rowDistance + colDistance;
    }
    // Left exit
    else if (Reader.hasLeftExit()) {
        int rowDistance = Math.abs(primary.getRow() - Reader.getExitRow());
        int colDistance = primary.getCol();
        return rowDistance + colDistance;
    }
    // Top exit
    else if (Reader.hasTopExit()) {
        int rowDistance = primary.getRow();
        int colDistance = Math.abs(primary.getCol() - Reader.getExitCol());
        return rowDistance + colDistance;
    }
    // Bottom exit
    else if (Reader.hasBottomExit()) {
        int endRow = primary.getRow() + (primary.isHorizontal() ? 0 : primary.getLength() - 1);
        int rowDistance = (state.getRows() - 1) - endRow;
        int colDistance = Math.abs(primary.getCol() - Reader.getExitCol());
        return rowDistance + colDistance;
    }
    return 0;
}
```

```

private static int blockingVehicles(State state) {
    List<Piece> pieces = state.getPieces();
    Piece primary = pieces.get(0);
    int blockingCount = 0;
    int distanceComponent = 0;

    // For right exit
    if (Reader.hasRightExit()) {
        int primaryRow = primary.getRow();
        int endCol = primary.getCol() + primary.getLength() - 1;

        // Count vehicles blocking the path between the primary piece and exit
        for (int i = 1; i < pieces.size(); i++) {
            Piece p = pieces.get(i);
            // Check if piece is in the way horizontally
            if (p.getRow() == primaryRow && p.getCol() > endCol) {
                blockingCount++;
            }
        }

        // Add distance component
        distanceComponent = Math.abs(primaryRow - Reader.getExitRow()) +
                           ((state.getCols() - 1) - endCol);
    }
    // For left exit
    else if (Reader.hasLeftExit()) {
        int primaryRow = primary.getRow();
        int startCol = primary.getCol();

        // Count vehicles blocking the path to the left
        for (int i = 1; i < pieces.size(); i++) {
            Piece p = pieces.get(i);
            // Check if piece is in the way horizontally
            if (p.getRow() == primaryRow && p.getCol() < startCol) {
                blockingCount++;
            }
        }

        // Add distance component
        distanceComponent = Math.abs(primaryRow - Reader.getExitRow()) + startCol;
    }
}

```

```

// For top exit
else if (Reader.hasTopExit()) {
    int primaryCol = primary.getCol();
    int startRow = primary.getRow();

    // Count vehicles blocking the path upward
    for (int i = 1; i < pieces.size(); i++) {
        Piece p = pieces.get(i);
        // Check if piece is in the way vertically
        if (p.getCol() == primaryCol && p.getRow() < startRow) {
            blockingCount++;
        }
    }

    // Add distance component
    distanceComponent = Math.abs(primaryCol - Reader.getExitCol()) + startRow;
}

// For bottom exit
else if (Reader.hasBottomExit()) {
    int primaryCol = primary.getCol();
    int endRow = primary.getRow() + (primary.isHorizontal() ? 0 : primary.getLength() - 1);

    // Count vehicles blocking the path downward
    for (int i = 1; i < pieces.size(); i++) {
        Piece p = pieces.get(i);
        // Check if piece is in the way vertically
        if (p.getCol() == primaryCol && p.getRow() > endRow) {
            blockingCount++;
        }
    }

    // Add distance component
    distanceComponent = Math.abs(primaryCol - Reader.getExitCol()) +
        ((state.getRows() - 1) - endRow);
}

// Weight the blocking count more heavily than distance
return (blockingCount * 2) + distanceComponent;
}
}

```

5. Main.java

```

1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4 import java.util.Scanner;
5 import java.util.Stack;
6
7 public class Main {
8     Run | Debug | Run main | Debug main
9     public static void main(String[] args) {
10         System.out.println("=====");
11         System.out.println("SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!");
12         System.out.println("=====");
13         System.out.println();
14
15         State goal = null;
16         Scanner scanner = new Scanner(System.in);
17         String filePath;
18
19         while (true) {
20             System.out.print("Masukkan path test case(.txt): ");
21             filePath = scanner.nextLine();
22             System.out.println("");
23             if (filePath.isEmpty()) {
24                 System.out.println("Path tidak boleh kosong! Coba lagi.");
25                 continue;
26             }
27             if (!Reader.readInputFile(filePath)) {
28                 System.out.println("Gagal membaca file test case atau file tidak ada.\n");
29                 continue;
30             } else {
31                 break;
32             }
33         }
34
35     public class Main {
36         public static void main(String[] args) {
37             System.out.println("=====");
38             System.out.println("Pilih algoritma pencarian:");
39             System.out.println("1. Greedy Best First Search");
40             System.out.println("2. Uniform Cost Search (UCS)");
41             System.out.println("3. A*");
42             System.out.println("Masukkan algoritma yang ingin digunakan: ");
43             String algorithmMethod;
44             while (true) {
45                 algorithmMethod = scanner.nextLine();
46                 if (algorithmMethod.isEmpty()) {
47                     System.out.println("Pilihan algoritma tidak boleh kosong! Coba lagi.");
48                     continue;
49                 }
50                 if (!algorithmMethod.equals(anObject:"1") && !algorithmMethod.equals(anObject:"2") && !algorithmMethod.equals(anObject:"3")) {
51                     System.out.println("Algoritma tidak valid.\n");
52                     continue;
53                 } else {
54                     break;
55                 }
56             }
57             int heuristicMethod = 0;
58             System.out.println("");
59             System.out.println("=====");
60             if (algorithmMethod.equals(anObject:"1") || algorithmMethod.equals(anObject:"3")) {
61                 System.out.println("Pilih Heuristik:");
62                 System.out.println("1. Manhattan Distance");
63                 System.out.println("2. Blocking Vehicles");
64                 while (true) {
65                     try {
66                         heuristicMethod = scanner.nextInt(); // Read integer input
67                         scanner.nextLine(); // Consume the leftover newline character
68                         if (heuristicMethod != 1 && heuristicMethod != 2) {
69                             System.out.println("Heuristik tidak valid.\n");
70                             continue;
71                         } else {
72                             break;
73                         }
74                     } catch (Exception e) {
75                         System.out.println("Input tidak valid! Masukkan angka 1 atau 2.\n");
76                         scanner.nextLine(); // Clear invalid input
77                     }
78                 }
79             }
80         }
81     }
82 }
```

```

7 public class Main {
8     public static void main(String[] args) {
9
10         System.out.println("=====");
11         System.out.println("Pilih algoritma pencarian:");
12         System.out.println("1. Greedy Best First Search");
13         System.out.println("2. Uniform Cost Search (UCS)");
14         System.out.println("3. A*");
15         System.out.println("Masukkan algoritma yang ingin digunakan: ");
16         String algorithmMethod;
17         while (true) {
18             algorithmMethod = scanner.nextLine();
19             if (algorithmMethod.isEmpty()) {
20                 System.out.println("Pilihan algoritma tidak boleh kosong! Coba lagi.");
21                 continue;
22             }
23             if (!algorithmMethod.equals(anObject:"1") && !algorithmMethod.equals(anObject:"2") && !algorithmMethod.equals(anObject:"3")) {
24                 System.out.println("Algoritma tidak valid.\n");
25                 continue;
26             } else {
27                 break;
28             }
29         }
30
31         int heuristicMethod = 0;
32         System.out.println("");
33         System.out.println("=====");
34         if (algorithmMethod.equals(anObject:"1") || algorithmMethod.equals(anObject:"3")) {
35             System.out.println("Pilih Heuristik:");
36             System.out.println("1. Manhattan Distance");
37             System.out.println("2. Blocking Vehicles");
38             while (true) {
39                 try {
40                     heuristicMethod = scanner.nextInt(); // Read integer input
41                     scanner.nextLine(); // Consume the leftover newline character
42                     if (heuristicMethod != 1 && heuristicMethod != 2) {
43                         System.out.println("Heuristik tidak valid.\n");
44                         continue;
45                     } else {
46                         break;
47                     }
48                 } catch (Exception e) {
49                     System.out.println("Input tidak valid! Masukkan angka 1 atau 2.\n");
50                     scanner.nextLine(); // Clear invalid input
51                 }
52             }
53         }
54     }
55 }
```

```

7 public class Main {
8     public static void main(String[] args) {
9
10         int heuristicMethod = 0;
11         System.out.println("");
12         System.out.println("=====");
13         if (algorithmMethod.equals(anObject:"1") || algorithmMethod.equals(anObject:"3")) {
14             System.out.println("Pilih Heuristik:");
15             System.out.println("1. Manhattan Distance");
16             System.out.println("2. Blocking Vehicles");
17             while (true) {
18                 try {
19                     heuristicMethod = scanner.nextInt(); // Read integer input
20                     scanner.nextLine(); // Consume the leftover newline character
21                     if (heuristicMethod != 1 && heuristicMethod != 2) {
22                         System.out.println("Heuristik tidak valid.\n");
23                         continue;
24                     } else {
25                         break;
26                     }
27                 } catch (Exception e) {
28                     System.out.println("Input tidak valid! Masukkan angka 1 atau 2.\n");
29                     scanner.nextLine(); // Clear invalid input
30                 }
31             }
32         }
33     }
34 }
```

```

7  public class Main {
8      public static void main(String[] args) {
9
10         long startTime = System.currentTimeMillis();
11         if (algorithmMethod.equals(anObject:"1")) {
12             System.out.println(x:"Menggunakan algoritma Greedy Best First Search.\n");
13             goal = Algorithm.greedy(Reader.getInitialState(), Reader.rows, Reader.cols, heuristicMethod);
14         } else if (algorithmMethod.equals(anObject:"2")) {
15             System.out.println(x:"Menggunakan algoritma Uniform Cost Search (UCS).\n");
16             goal = Algorithm.ucs(Reader.getInitialState(), Reader.rows, Reader.cols);
17         } else {
18             System.out.println(x:"Menggunakan algoritma A*.\n");
19             goal = Algorithm.aStar(Reader.getInitialState(), Reader.rows, Reader.cols, heuristicMethod);
20         }
21         if (goal != null) {
22             System.out.println(x:"Solution found!");
23             printsolutionPath(goal);
24         } else {
25             System.out.println(x:"No solution found.");
26         }
27
28         long endTime = System.currentTimeMillis();
29
30         // Output
31         System.out.println("Banyak gerakan yang diperiksa: " + Algorithm.getNodesVisited() +"\n");
32         System.out.println("Waktu pencarian: " + (endTime - startTime) + " ms\n");
33
34
35     public class Main {
36         public static void main(String[] args) {
37
38             if (goal != null) {
39                 // Prompt menyimpan hasil
40                 System.out.println(x:"Apakah anda ingin menyimpan hasilnya? (ya/tidak)");
41                 String save_solusi = scanner.nextLine();
42                 if (save_solusi.equals(anObject:"ya")) {
43                     System.out.println(x:"Masukkan nama file untuk menyimpan hasil: ");
44                     String output_filename = scanner.nextLine();
45                     outputFile(goal, "test/solution/" + output_filename + ".txt");
46                     System.out.println(x:"");
47                     System.out.println(x:"Solusi berhasil disimpan pada test/solution!");
48                 }
49                 System.out.println();
50             }
51             scanner.close();
52         }
53
54     public class Main {
55
56         public static void printsolutionPath(State state) {
57
58             Stack<State> path = new Stack<>();
59             while (state != null) {
60                 path.push(state);
61                 state = state.getParent();
62             }
63
64             int step = 1;
65             while (!path.isEmpty()) {
66                 State s = path.pop();
67                 if (s.getParent() != null) {
68                     System.out.print("Gerakan " + step + ": ");
69                     System.out.println(s.getMovedPiece() + " - " + s.getMovedPieceDirection());
70                     step++;
71                 } else {
72                     System.out.println(x:"Papan awal: ");
73                 }
74             }
75             System.out.println(s.getBoard(Reader.rows, Reader.cols));
76         }
77     }
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139

```

```
7  public class Main {
139
140    public static void outputFile(State state, String filenames){
141        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filenames))) {
142            Stack<State> path = new Stack<>();
143            while (state != null) {
144                path.push(state);
145                state = state.getParent();
146            }
147
148            int step = 1;
149            while (!path.isEmpty()) {
150                State s = path.pop();
151                if (s.getParent() != null) {
152                    writer.write("Gerakan " + step + ": ");
153                    writer.write(s.getMovedPiece() + " - " + s.getMovedPieceDirection());
154                    writer.newLine();
155                    step++;
156                } else {
157                    writer.write(str:"Papan awal: ");
158                    writer.newLine();
159                }
160
161                writer.write(s.getBoardPlain(Reader.rows, Reader.cols));
162                writer.newLine();
163            }
164        } catch (IOException e){
165            System.out.println(x:"error!");
166            e.printStackTrace();
167        }
168    }
169
170
171 }
```

BAB IV

ANALISIS DAN PENGUJIAN

4.1. Analisis Kompleksitas Algoritma Program

4.1.1. Analisis Kompleksitas Algoritma *Uniform Cost Search* (UCS)

Algoritma *Uniform Cost Search* (UCS) menggunakan $g(n)$ sebagai fungsi yang menyatakan *cost* jalur dari *root* sampai *node* ke-n. Kompleksitas waktu UCS bergantung pada jumlah *node* dan bagaimana *node* dieksplorasi berdasarkan biaya lintasan. Operasi utama algoritma memasukkan tetangga dari *node* ke dalam *priority queue*, semua *node* diekstrak, dan semua *node* dan sisi diproses dalam kasus terburuk. Oleh karena itu, kompleksitas waktu Algoritma UCS adalah $O((V+E) * \log V)$ dengan V sebagai *node* dan E sebagai *edge*.

Kompleksitas ruang algoritma *Uniform Cost Search* (UCS) bergantung pada jumlah simpul yang disimpan dalam *priority queue* dan himpunan simpul yang telah dikunjungi. Kompleksitas ruang algoritma UCS adalah $O(V)$ dengan V sebagai *node*.

4.1.2. Analisis Kompleksitas Algoritma *Greedy Best First Search*

Algoritma *Greedy Best First Search* menggunakan fungsi heuristik $h(n)$ untuk memandu eksplorasi *node*. Algoritma ini akan selalu memilih *node* dengan nilai heuristik paling rendah. Operasi utama algoritma memasukkan tetangga dari *node* ke dalam *priority queue*, semua *node* diekstrak, dan semua *node* dan sisi diproses dalam kasus terburuk. Oleh karena itu, kompleksitas waktu Algoritma *Greedy Best First Search* adalah $O((V+E) * \log V)$ dengan V sebagai *node* dan E sebagai *edge*.

Kompleksitas ruang algoritma *Greedy Best First Search* bergantung pada jumlah simpul yang disimpan dalam *priority queue* dan himpunan simpul yang telah dikunjungi. Kompleksitas ruang algoritma *Greedy Best First Search* adalah $O(V)$ dengan V sebagai *node*.

4.1.3. Analisis Kompleksitas Algoritma A*

Algoritma A* menggunakan fungsi biaya $f(n)$ total dari heuristik $h(n)$ dan $g(n)$ seperti yang disebut pada algoritma *Uniform Cost Search*. A* mencari lintasan optimal dengan bantuan heuristik yang konsisten dan tidak melebih-lebihkan. Operasi utama algoritma memasukkan tetangga dari *node* ke dalam *priority queue*, semua *node* diekstrak, dan semua *node* dan sisi diproses dalam kasus terburuk. Oleh karena itu, kompleksitas waktu Algoritma A* adalah $O((V+E) * \log V)$ dengan V sebagai *node* dan E sebagai *edge*.

Kompleksitas ruang algoritma A* bergantung pada jumlah simpul yang disimpan dalam *priority queue* dan himpunan simpul yang telah dikunjungi. Kompleksitas ruang algoritma A* adalah O(V) dengan V sebagai *node*.

4.2. Hasil dan Analisis Percobaan

4.2.1. Konfigurasi Puzzle Sama, Algoritma Berbeda

1. Test Case 1

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test1.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
2

=====
Menggunakan algoritma Uniform Cost Search (UCS).
```

- Output

```
Solution found!
Papan awal:
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 1: C - Atas
AAB.C.F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 2: P - Kanan
AABC.F
..BCDF
G.PPDFK
GH.III
GHJ...
LLJMM.

Gerakan 3: I - Kiri
AABC.F
..BCDF
G.PPDFK
GH.III.
GHJ...
LLJMM.

=====
G.PP..K
GHIIIF
GHJ..F
LLJMMF

Gerakan 8: P - Kanan
AABCD.
..BCD.
G..PP.K
GHIIIF
GHJ..F
LLJMMF

Gerakan 9: P - Kanan
AABCD.
..BCD.
G...PPK
GHIIIF
GHJ..F
LLJMMF

Banyak gerakan yang diperiksa: 666

Waktu pencarian: 88 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi1_ucs

Solusi berhasil disimpan pada test/solution!
```

2. Test Case 2

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test1.txt
=====

Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
1

=====

Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
1
Menggunakan algoritma Greedy Best First Search.
```

- Output

```
Solution found!
Papan awal:
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 1: C - Atas
AAB..F
..BCDF
GPPDFK
GH.III
GHJ...
LLJMM.

Gerakan 2: P - Kanan
AABC.F
..BCDF
G.PPDFK
GH.III
GHJ...
LLJMM.

Gerakan 3: D - Atas
AABCD.F
..BCDF
G.PPFK
GH.III
GHJ...
LLJMM.
```

```
..BCD.
GHBPPFK
GHIIIF
G.J..F
LLJMM.

Gerakan 85: F - Bawah
.AACD.
..BCD.
GHBPPFK
GHIIIF
G.J..F
LLJMMF

Gerakan 86: P - Kanan
.AACD.
..BCD.
GHB.PPK
GHIIIF
G.J..F
LLJMMF

Banyak gerakan yang diperiksa: 273
Waktu pencarian: 126 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi1_greedy_heuristic1

Solusi berhasil disimpan pada test/solution!
```

3. Test Case 3

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test1.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
1

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
2
```

- Output

<pre>G.PPFK G.III G.J... LLJ.MM Gerakan 7: M - Kiri AABCDF ..BCDF GH.PPFK GH.III G.J... LLJ.MM. Gerakan 8: J - Atas AABCDF ..BCDF GH.PPFK GH.III G.J... LLJ.MM. Gerakan 9: M - Kanan AABCDF ..BCDF GH.PPFK GH.III G.J... LLJ.MM. Gerakan 10: L - Kanan AABCDF ..BCDF GH.PPFK</pre>	<pre>G.BPPFK GHIII.F .HJ...F LLJMM. Gerakan 132: F - Bawah AA.CD. G.BCD. G.BPPFK GHIII.F .HJ...F LLJMMF Gerakan 133: P - Kanan AA.CD. G.BCD. G.B.PPK GHIII.F .HJ...F LLJMMF Banyak gerakan yang diperiksa: 588 Waktu pencarian: 243 ms Apakah anda ingin menyimpan hasilnya? (ya/tidak) ya Masukkan nama file untuk menyimpan hasil: solusi1_greedy_heuristic2 Solusi berhasil disimpan pada test/solution!</pre>
--	---

4. Test Case 4

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test1.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
3

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
1
Menggunakan algoritma A*.
```

- Output

```
Solution found!
Papan awal:
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 1: D - Atas
AAB.DF
..BCDF
GPPC.FK
GH.III
GHJ...
LLJMM.

Gerakan 2: I - Kiri
AAB.DF
..BCDF
GPPC.FK
GH.III.
GHJ...
LLJMM.

Gerakan 3: C - Atas
AAB.CDF
..BCDF
GPPC.FK
GH.III.
GHJ...
LLJMM.

=====
G..PPFK
GHIIIF
GHJ..F
LLJMM.

Gerakan 8: F - Bawah
AABCD.
..BCD.
G..PPFK
GHIIIF
GHJ..F
LLJMMF

Gerakan 9: P - Kanan
AABCD.
..BCD.
G..PPFK
GHIIIF
GHJ..F
LLJMMF

Banyak gerakan yang diperiksa: 429

Waktu pencarian: 88 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi1_astar_heuristic1

Solusi berhasil disimpan pada test/solution!
```

5. Test Case 5

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test1.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
3

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
2
Menggunakan algoritma A*.
```

- Output

```
Solution found!
Papan awal:
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 1: I - Kiri
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 2: F - Bawah
AAB...
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 3: F - Bawah
AAB...
..BCDF
GPPCDFK
GH.III
GHJ..F
LLJMM.

..BCD.
G..PP..K
GH.III
GHJ..F
LLJMMF

Gerakan 8: P - Kanan
AABCD.
..BCD.
G..PP.K
GH.III
GHJ..F
LLJMMF

Gerakan 9: P - Kanan
AABCD.
..BCD.
G..PPK
GH.III
GHJ..F
LLJMMF

Banyak gerakan yang diperiksa: 416
Waktu pencarian: 82 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi1_astar_heuristic2

Solusi berhasil disimpan pada test/solution!
```

Analisis

Untuk konfigurasi puzzle test1.txt, semua algoritma berhasil menemukan solusi. Algoritma UCS dan A* menghasilkan solusi dengan gerakan paling sedikit, sedangkan algoritma GBFS membutuhkan gerakan yang lebih banyak. Algoritma GBFS menghasilkan solusi dengan gerakan yang lebih banyak karena hanya berfokus pada nilai heuristik yang terkecil dan tidak menyadari bahwa sebenarnya butuh gerakan yang lebih banyak. Hal ini menyebabkan waktu pencarian algoritma UCS dan A* juga lebih cepat dibandingkan algoritma GBFS. Gerakan (*node*) yang diperiksa algoritma A* juga lebih

sedikit daripada yang diperiksa algoritma UCS karena ada tambahan nilai heuristik yang dapat mempengaruhi ekspansi *node*.

4.2.2. Konfigurasi Puzzle Berbeda, Algoritma Uniform Cost Search

1. Test Case 6

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test2.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
2

=====
Menggunakan algoritma Uniform Cost Search (UCS).
```

- Output

<pre>Solution found! Papan awal: A...BB ACC... ..D..E PPDF.EK GHHF.I GJILLI Gerakan 1: C - Kanan A...BB A.CC.. ..D..E PPDF.EK GHHF.I GJILLI Gerakan 2: C - Kanan A...BB A..CC. ..D..E PPDF.EK GHHF.I GJILLI Gerakan 3: C - Kanan A...BB A...CC ..D..E PPDF.EK GHHF.I GJILLI</pre>	<pre>ACCF.E G..F.E G..PP..K HHD..I JJDLII Gerakan 26: P - Kanan A...BB ACCF.E G..F.E G..PP.K HHD..I JJDLII Gerakan 27: P - Kanan A...BB ACCF.E G..F.E G..PPK HHD..I JJDLII Banyak gerakan yang diperiksa: 11206 Waktu pencarian: 301 ms Apakah anda ingin menyimpan hasilnya? (ya/tidak) ya Masukkan nama file untuk menyimpan hasil: solusi2_ucs Solusi berhasil disimpan pada test/solution!</pre>
--	--

2. Test Case 7

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test3.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
2

=====
Menggunakan algoritma Uniform Cost Search (UCS).
```

- Output

```
Solution found!
Papan awal:
    K
A...BB
ACC...
..D..P
..DFEP
GHHF..
GJLL.

Gerakan 1: B - Kiri
    K
A..BB.
ACC...
..D..P
..DFEP
GHHF..
GJLL.

Gerakan 2: P - Atas
    K
A..BB.
ACC..P
..D..P
..DFE.
GHHF..
GJLL.

Gerakan 3: P - Atas
    K
A..BBP
ACC..P
..D...
..DFE.
GHHF..
GJLL.

Banyak gerakan yang diperiksa: 257
Waktu pencarian: 64 ms
Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi3_ucs
Solusi berhasil disimpan pada test/solution!
```

3. Test Case 8

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test4.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
2

=====
Menggunakan algoritma Uniform Cost Search (UCS).
```

- Output

```

Solution found!
Papan awal:
.AABB.
.CDE..
.FDEGG
KHFPPIJ
HFL.IJ
H.LMMJ

Gerakan 1: B - Kanan
.AA,BB
CDE..
.FDEGG
KHFPPIJ
HFL.IJ
H.LMMJ

Gerakan 2: E - Atas
.AA,BB
CDE..
.FDGG
KHFPPIJ
HFL.IJ
H.LMMJ

Gerakan 3: G - Kiri
.AA,BB
CDE..
.FDGG
KHFPPIJ
HFL.IJ
H.LMMJ

Gerakan 4: H - Atas
.HFABBB
.FDCIJ
.HFDGGJ
K.PP..J
..LEI.
MMLIEI.

Gerakan 51: H - Atas
.HFAABB
.HFDCCJ
.HFDGGJ
K.PP..J
..LEI.
MMLIEI.

Gerakan 52: P - Kiri
.HFAABB
.HFDCCJ
.HFDGGJ
K.PP..J
..LEI.
MMLIEI.

Banyak gerakan yang diperiksa: 5242
Waktu pencarian: 214 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi4_ucs

Solusi berhasil disimpan pada test/solution!

```

4. Test Case 9

- Input

```

=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test5.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
2

=====
Menggunakan algoritma Uniform Cost Search (UCS).

```

- Output

```

Solution found!
Papan awal:
.IIGP..
.JGP..
.J.CC.
.JEEAA
.FFF.B
.HHDBB
K

Gerakan 1: C - Kanan
.IIGP..
.JGP..
.J..CC
.JEEAA
.FFF.B
.HHDBB
K

Gerakan 2: P - Bawah
.IIG...
.JGP..
.J.PCC
.JEEAA
.FFF.B
.HHDBB
K

Gerakan 3: G - Bawah
.II...
.JGP..
.JGCC
.JEEAA

FFF...
.HH.DD
K

Gerakan 53: P - Bawah
.JII.B
.JG..B
.JGCC
.EEAA
.FFF...
.HH.DD
K

Gerakan 54: P - Bawah
.JII.B
.JG..B
.JGCC
.EEAA
.FFF...
.HH.DD
K

Banyak gerakan yang diperiksa: 3847
Waktu pencarian: 203 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi5_ucs

Solusi berhasil disimpan pada test/solution!

```

Analisis

Untuk menguji algoritma UCS, digunakan test2.txt, tes3.txt, test4.txt, dan test5.txt dengan masing-masing konfigurasi memiliki *exit* pada letak berbeda-beda (kanan, atas, kiri, bawah). Gerakan (*node*) yang diperiksa oleh algoritma UCS jauh lebih banyak daripada gerakan yang dibutuhkan pada solusi. Hal ini disebabkan oleh algoritma UCS yang merupakan *Blind Search* melakukan ekspansi *node* hanya berdasarkan *cost* gerakan dari awal konfigurasi ke *state* ke-n tanpa mengetahui seberapa dekat gerakan menuju *goal*. Seperti algoritma BFS, algoritma UCS menelusuri semua kemungkinan gerakan yang mungkin. Berdasarkan percobaan, waktu pencarian dengan algoritma UCS semakin meningkat seiring bertambahnya jumlah piece. Solusi yang diberikan pasti optimal atau memberikan gerakan paling sedikit.

4.2.3. Konfigurasi Puzzle Berbeda, Algoritma Greedy Best First Search

1. Test Case 10
 - Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test4.txt
=====

Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
1

=====

Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
1
```

- Output

<p>Gerakan 139: G - Kanan</p> <pre>AA.EBB CCDE.. HFDGG KHF.PPJ HFL.IJ MML.IJ</pre> <p>Gerakan 140: E - Bawah</p> <pre>AA..BB CCDE.. HFDGG KHF.PPJ HFL.IJ MML.IJ</pre> <p>Gerakan 141: L - Atas</p> <pre>AA..BB CCDE.. HFDGG KHF.LPPJ HFL.IJ MML.IJ</pre> <p>Gerakan 142: M - Kanan</p> <pre>AA..BB CCDE.. HFDGG KHF.LPPJ HFL.IJ MML.IJ</pre> <p>Gerakan 143: M - Kanan</p>	<pre>HFDCC. HFDGG K.PP..J ..LEIJ MMLEI.</pre> <p>Gerakan 262: H - Atas</p> <pre>HFAAB HFDCC. HFDGG K.PP..J ..LEIJ MMLEI.</pre> <p>Gerakan 263: P - Kiri</p> <pre>HFAAB HFDCC. HFDGG KPP...J ..LEIJ MMLEI.</pre> <p>Banyak gerakan yang diperiksa: 1929</p> <p>Waktu pencarian: 288 ms</p> <p>Apakah anda ingin menyimpan hasilnya? (ya/tidak) ya</p> <p>Masukkan nama file untuk menyimpan hasil: solusi4_greedy_heuristic1</p> <p>Solusi berhasil disimpan pada test/solution!</p>
--	---

2. Test Case 11

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test5.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
1

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
1
```

- Output

<p>Gerakan 659: J - Atas</p> <pre>.J.GPII .JGP.B .J.CCB .J.EEAA ..FFF. ..HHDD K</pre> <p>Gerakan 660: H - Kiri</p> <pre>.J.GPII .JGP.B .J.CCB ..EEAA ..FFF. .HH.DD K</pre> <p>Gerakan 661: J - Bawah</p> <pre>.GPII .JGP.B .J.CCB .J.EEAA ..FFF. ..HH.DD K</pre> <p>Gerakan 662: J - Bawah</p> <pre>.GPII .JGP.B .J.CCB .J.EEAA ..FFF.</pre>	<p>FFF.K.. ..HHDD K</p> <p>Gerakan 770: H - Kiri</p> <pre>.J.IIB .JG..B .JG.CC .EEPAA .FFFP.. .HH.DD K</pre> <p>Gerakan 771: P - Bawah</p> <pre>.J.IIB .JG..B .JG.CC .EEAA .FFFP.. ..HHDD K</pre> <p>Banyak gerakan yang diperiksa: 2797</p> <p>Waktu pencarian: 552 ms</p> <p>Apakah anda ingin menyimpan hasilnya? (ya/tidak) ya</p> <p>Masukkan nama file untuk menyimpan hasil: solusi5_greedy_heuristic1</p> <p>Solusi berhasil disimpan pada test/solution!</p>
--	---

3. Test Case 12

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test4.txt
=====

Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
1

=====

Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
2
```

- Output

```
Solution found!
Papan awal:
.AABB.
.CCDE..
.FDEGG
KHFPIJ
.HFL.IJ
.H.LMMJ

Gerakan 1: H - Atas
.AABB.
.CCDE..
.FDEGG
KHFPIJ
.HFL.IJ
..LMMJ

Gerakan 2: B - Kanan
.AA..BB
.CCDE..
.HFDEGG
KHFPIJ
.HFL.IJ
..LMMJ

Gerakan 3: A - Kiri
AA..BB
.CCDE..
.HFDEGG
KHFPIJ
.HFL.IJ
..LMMJ

HFD.GG
KHPP.IJ
..LEIJ
MMLE.J

Gerakan 94: H - Atas
FAABB
.FD.CC
.FD.GG
KPP.IJ
..LEIJ
MMLE.J

Gerakan 95: P - Kiri
HFAABB
.HFD.CC
.HFD.GG
KPP.IJ
..LEIJ
MMLE.J

Banyak gerakan yang diperiksa: 2204

Waktu pencarian: 196 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi4_greedy_heuristic2

Solusi berhasil disimpan pada test/solution!
```

4. Test Case 13

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test5.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
1

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
2
```

- Output

The screenshot shows the application's interface for solving a Rush Hour puzzle. It displays the initial state of the puzzle board, a sequence of moves, and the final solution.

Solution found!

Papan awal:

```
IIGP...
.JGP...
.J.CC.
.JEAA
.FFF.B
.HHDBB
K
```

Gerakan 1: C - Kanan

```
IIGP...
.JGP...
.J.CC
.JEAA
.FFF.B
.HHDBB
K
```

Gerakan 2: P - Bawah

```
IIG...
.JGP...
.J.PCC
.JEAA
.FFF.B
.HHDBB
K
```

Gerakan 3: H - Kiri

```
IIG...
.JGP...
.J.PCC
.JEAA
```

Final State (After Solution):

```
FFF...
HH..DD
K
```

Gerakan 66: P - Bawah

```
.JII.B
.JG..B
.JG.CC
.EEPAA
.FFP...
HH..DD
K
```

Gerakan 67: P - Bawah

```
.JII.B
.JG..B
.JG.CC
.EEAA
.FFP...
HH.PDD
K
```

Banyak gerakan yang diperiksa: 1449

Waktu pencarian: 169 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya

Masukkan nama file untuk menyimpan hasil:
solusi5_greedy_heuristic2

Solusi berhasil disimpan pada test/solution!

Analisis

Untuk menguji algoritma GBFS, digunakan test4.txt dan test5.txt, keduanya digunakan untuk menguji dan membandingkan hasil dari penggunaan heuristik yang berbeda. Gerakan (*node*) yang diperiksa dan solusi yang dihasilkan oleh algoritma GBFS dengan kedua heuristik menunjukkan jumlah yang berbeda. Hal ini disebabkan heuristik Manhattan Distance hanya memperhitungkan jarak *primary piece* ke *exit*, sedangkan heuristik Blocking Vehicles hanya mempertimbangkan mobil penghalang antara *primary piece* dengan *exit* sehingga dapat mempengaruhi ekspansi *node*. Perbedaan solusi dengan kedua heuristik tersebut menunjukkan bahwa algoritma GBFS tidak menjamin solusi yang optimal.

4.2.4. Konfigurasi Puzzle Berbeda, Algoritma A*

1. Test Case 14

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test6.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
3

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
1
```

- Output

```
Solution found!
Papan awal:
.ABCCC
.AB..D
.APPEDK
FF.GED
...GHH
.....
Gerakan 1: G - Bawah
.ABCCC
.AB..D
.APPEDK
FF..ED
...GHH
...G..
Gerakan 2: F - Kanan
.ABCCC
.AB..D
.APPEDK
.FF..ED
...GHH
...G..
Gerakan 3: F - Kanan
.ABCCC
.AB..D
.APPEDK
..FFED
...GHH
...G..
```

```
...GE
...PRK
.ABFBD
.ABHHD
.A.....
Gerakan 31: D - Bawah
CCCGE.
...GE.
...PRK
.ABFBD
.ABHHD
.A...D
Gerakan 32: P - Kanan
CCCGE.
...GE.
...PRK
.ABFBD
.ABHHD
.A...D
Banyak gerakan yang diperiksa: 1557
Waktu pencarian: 124 ms
Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi6_astar_heuristic1
Solusi berhasil disimpan pada test/solution!
```

2. Test Case 15

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test7.txt

=====
Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
3

=====
Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
1
```

- Output

```
Solution found!
Papan awal:
K
CCCE.H
B.PE.H
B.PDDD
AAA.G.
...G.
...FF

Gerakan 1: A - Kanan
K
CCCE.H
B.PE.H
B.PDDD
.AAAG.
...G.
...FF

Gerakan 2: F - Kiri
K
CCCE.H
B.PE.H
B.PDDD
.AAAG.
...G.
...FF

Gerakan 3: F - Kiri
K
CCCE.H
B.PE.H
B.PDDD

...AAA
...EGH
FF.EGH

Gerakan 82: C - Kanan
K
B..CCC
B.P...
...PDDD
...AAA
...EGH
FF.EGH

Gerakan 83: P - Atas
K
B.PCCC
B.P...
...PDDD
...AAA
...EGH
FF.EGH

Banyak gerakan yang diperiksa: 8565
Waktu pencarian: 269 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya
Masukkan nama file untuk menyimpan hasil:
solusi7_astar_heuristic1

Solusi berhasil disimpan pada test/solution!
```

3. Test Case 16

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test6.txt
=====

Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
3

=====

Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
2
```

- Output

```
Solution found!
Papan awal:
.ABCC
.AB..D
.APPEDK
FF.GED
...GHH
.....
Gerakan 1: E - Atas
.ABCCC
.AB..ED
.APPEDK
FF.GED
...GHH
.....
Gerakan 2: F - Kanan
.ABCCC
.AB..ED
.APPEDK
.FFG.D
...GHH
.....
Gerakan 3: G - Bawah
.ABCCC
.AB..ED
.APPEDK
.FF.D
...GHH
...G..
```

...GE.
.PP..K
.ABFFD
.ABHHD
.A...D

Gerakan 31: P - Kanan
CCCGE.
...GF.
...PP.K
.ABFFD
.ABHHD
.A...D

Gerakan 32: P - Kanan
CCCGE.
...GE.
...PPK
.ABFFD
.ABHHD
.A...D

Banyak gerakan yang diperiksa: 1487

Waktu pencarian: 136 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya

Masukkan nama file untuk menyimpan hasil:
solusi6_astar_heuristic2

Solusi berhasil disimpan pada test/solution!

4. Test Case 17

- Input

```
=====
SELAMAT DATANG DI RUSH HOUR PUZZLE SOLVER!
=====

Masukkan path test case(.txt): D:\Lenovo\Documents\Stima\Tucil3_13523060_13523070\test\test7.txt
=====

Pilih algoritma pencarian:
1. Greedy Best First Search
2. Uniform Cost Search (UCS)
3. A*
Masukkan algoritma yang ingin digunakan:
3

=====

Pilih Heuristik:
1. Manhattan Distance
2. Blocking Vehicles
2
```

- Output

```
Solution found!
Papan awal:
K
CCCE.H
B.PE.H
B.PDDD
AAA.G.
....G.
....FF

Gerakan 1: A - Kanan
K
CCCE.H
B.PE.H
B.PDDD
.AAAG.
....G.
....FF

Gerakan 2: F - Kiri
K
CCCE.H
B.PE.H
B.PDDD
.AAAG.
....G.
....FF

Gerakan 3: F - Kiri
K
CCCE.H
B.PE.H
B.PDDD
```

..PDDD
..PAAA
..EGH
FF.EGH

Gerakan 82: P - Atas
K
B..CCC
B..P...
..PDDD
...AAA
...EGH
FF.EGH

Gerakan 83: P - Atas
K
B.PCCC
B.P...
...DDD
...AAA
...EGH
FF.EGH

Banyak gerakan yang diperiksa: 8410

Waktu pencarian: 369 ms

Apakah anda ingin menyimpan hasilnya? (ya/tidak)
ya

Masukkan nama file untuk menyimpan hasil:
solusi7_astar_heuristic2

Solusi berhasil disimpan pada test/solution!

Analisis

Untuk menguji algoritma A*, digunakan test6.txt dan test7.txt, keduanya digunakan untuk menguji dan membandingkan hasil dari penggunaan heuristik yang berbeda. Gerakan (*node*) yang diperiksa dan gerakan pada solusi yang dihasilkan oleh algoritma A* untuk tiap heuristik menunjukkan jumlah yang sama. Hal ini karena algoritma A* tidak hanya mempertimbangkan nilai heuristik melainkan juga *cost* atau banyak gerakan dari konfigurasi awal menuju state ke-n sehingga ekspansi *node* berbeda daripada algoritma GBFS meskipun sama-sama menggunakan heuristik. Selain itu, karena $h(n)$ yang digunakan tidak lebih besar daripada $h^*(n)$, maka algoritma A* menjamin solusi yang optimal.

BAB V

IMPLEMENTASI BONUS

5.1. Heuristic Alternatif

Total heuristik yang diimplementasikan berjumlah 2 yakni *Manhattan Distance* dan *Blocking Vehicles*. *Manhattan Distance* sendiri merupakan metode pengukuran jarak antara dua titik dengan menjumlahkan perbedaan absolut dari koordinat-koordinatnya. Beda dengan jarak *Euclidean*, *Manhattan Distance* menghitung jarak sepanjang sumbu yang sejajar dengan sumbu koordinat dan tidak langsung diagonal seperti *Euclidean*. Dalam kasus ini, dua titik yang jaraknya dicari ialah jarak dari *primary piece* menuju jalan keluar.

Blocking Vehicles adalah heuristik yang menghitung jumlah penghalang atau kendaraan yang menghalangi jalur di antara dua titik. Selain menghitung jumlah penghalang, heuristik ini juga menghitung *Manhattan Distance* sebagai komponen jarak namun bobot lebih besar ditetapkan pada jumlah penghalang. Dalam kasus ini, dua titik yang penghalangnya dihitung ialah dari *primary piece* menuju jalan keluar.

BAB VI

KESIMPULAN

Berdasarkan analisis dan implementasi algoritma *pathfinding* yakni algoritma *Uniform Cost Search*, algoritma *Greedy Best First Search*, dan algoritma A* pada penyelesaian puzzle Rush Hour, dapat disimpulkan bahwa masing-masing algoritma beserta heuristik yang digunakan efektif dalam menemukan solusi yakni gerakan-gerakan yang diperlukan untuk memindahkan *primary piece* ke *exit* dan menyelesaikan puzzle. Tiap algoritma menggunakan pertimbangan yang berbeda dalam ekspansi *node* pencarian jalur. Algoritma *Uniform Cost Search* (UCS) menggunakan nilai $g(n)$ atau *cost* dari konfigurasi awal menuju state ke-n, algoritma *Greedy Best First Search* menggunakan nilai $h(n)$ atau heuristik yakni Manhattan Distance dan Blocking Distance, dan algoritma A* menggunakan nilai $f(n) = g(n) + h(n)$ atau kombinasi keduanya. Dari hasil percobaan, didapatkan bahwa masing-masing algoritma memiliki kelebihan dan kekurangan masing-masing. Algoritma UCS dapat menjamin solusi optimal tetapi dapat lebih lambat karena tidak ada informasi tambahan seperti heuristik. Algoritma GBFS dapat mencari solusi dengan cepat tetapi dapat terjebak dalam jalur yang tampak memiliki heuristik dengan nilai kecil tapi sebenarnya memerlukan banyak langkah sehingga tidak menjamin solusi yang optimal. Algoritma A* mengambil kebaikan dari algoritma UCS dan algoritma GBFS sehingga menjamin solusi yang optimal (selama heuristiknya admissible) dan lebih cepat. Melalui tugas kecil ini, kami dapat lebih memahami baik secara teori maupun implementasi dari algoritma *pathfinding* yang telah diajarkan pada perkuliahan.

LAMPIRAN

Repository Github:

https://github.com/angelinaefrina/Tucil3_13523060_13523070.git

Tabel Pemeriksaan:

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	✓	

REFERENSI

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf)

<https://soc.sbinus.ac.id/2013/04/23/uninformed-search-dan-informed-search/>

<https://soc.sbinus.ac.id/2017/08/24/searching-uniform-cost-search/>

<https://www.kodeco.com/3016-introduction-to-a-pathfinding>

<https://www.graphable.ai/blog/pathfinding-algorithms/#:~:text=The%20algorithm%20starts%20by%20evaluating,G%20is%20the%20path%20A%2DB%2DD%2DE%2DG.>