LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA

Disusun oleh:

Boye Mangaratua Ginting
13523127

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2025

DAFTAR ISI

BAB 1 DESKRIPSI TUGAS

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

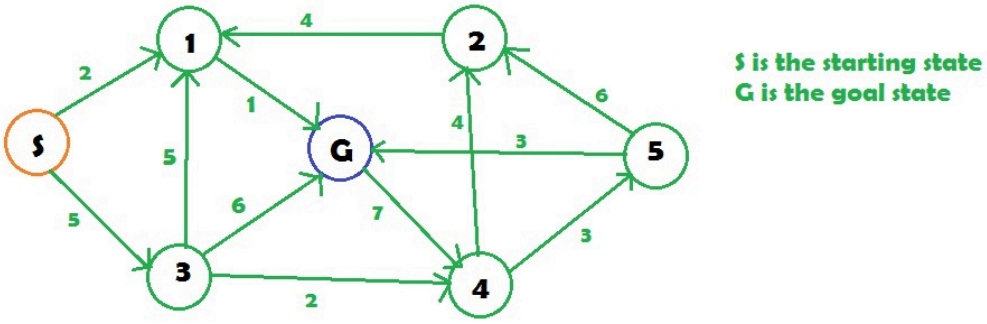
5. **Gerakan** — *Gerakan* yang dimaksudkan adalah pergeseran *piece* di dalam permainan. *Piece* hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

BAB 2 LANDASAN TEORI

2.1 Algoritma PathFinding

2.1.1 Algoritma Uniform Cost Search

Uniform Cost Search, juga dikenal sebagai variasi dari algoritma Dijkstra untuk graf berbobot, adalah algoritma pencarian buta yang mengeksplorasi node berdasarkan biaya jalur kumulatif terendah dari node awal ke node tersebut. UCS menggunakan antrian prioritas untuk menyimpan node, di mana prioritas ditentukan oleh biaya g(n) (total biaya dari awal).



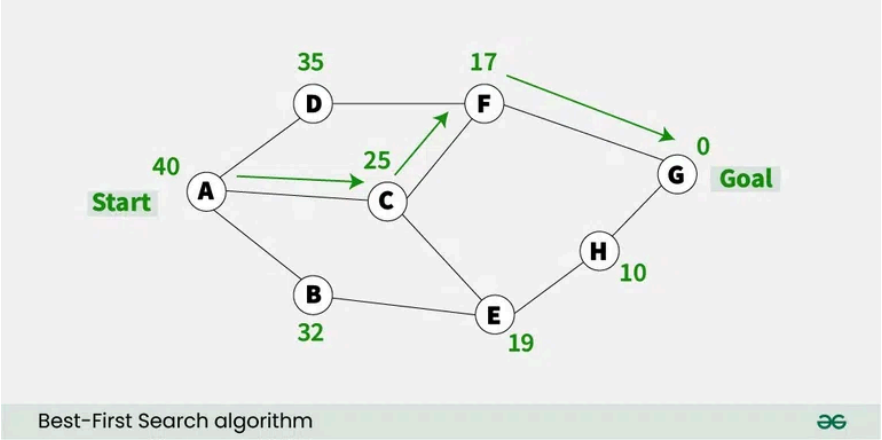
Gambar 2.1.1.1. Graf Uniform Cost Search

(Sumber : https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/)

Algoritma ini menjamin solusi optimal jika semua biaya edge non-negatif, menjadikannya cocok untuk masalah seperti routing jaringan atau optimasi transportasi. Prosesnya melibatkan pengeluaran node dengan biaya terendah, pengecekan apakah itu tujuan, dan penambahan node tetangga ke antrian dengan biaya yang diperbarui, hingga tujuan tercapai atau semua node dieksplorasi.

2.1.2 Algoritma Greedy Best-First Search

Greedy Best-First Search adalah algoritma pencarian berbasis heuristik yang memilih langkah berikutnya berdasarkan perkiraan jarak terdekat ke tujuan, diukur oleh fungsi heuristik h(n).



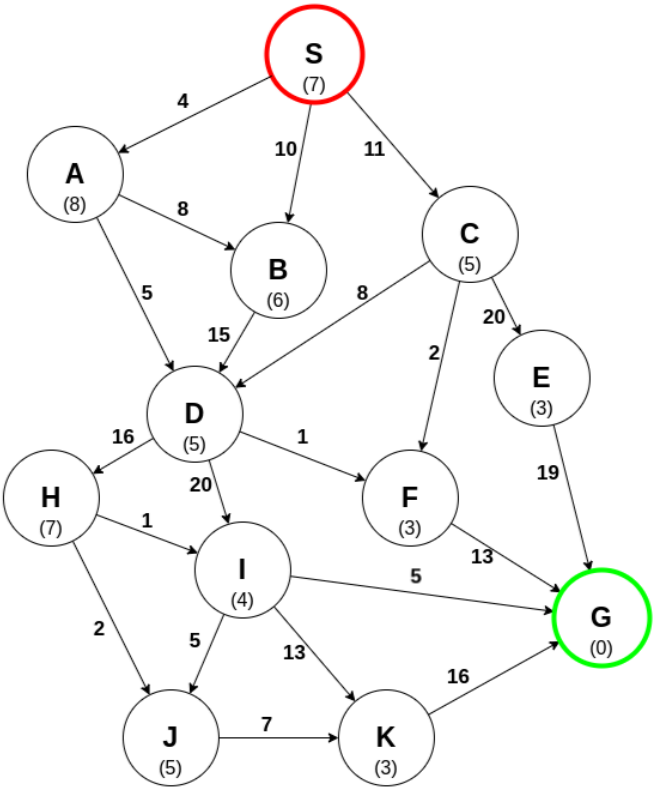
Gambar 2.1.2.1. Graf Greedy Best-First Search

(Sumber : https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/)

Algoritma ini bersifat "serakah" karena selalu memilih node dengan nilai heuristik terendah tanpa mempertimbangkan biaya jalur yang telah ditempuh. GBFS sering digunakan dalam masalah seperti perencanaan rute atau pencarian jalur di peta, tetapi tidak menjamin solusi optimal karena fokusnya hanya pada langkah saat ini. Kekurangannya adalah bisa terjebak pada jalur lokal yang tidak mengarah ke solusi global jika heuristiknya tidak cukup akurat.

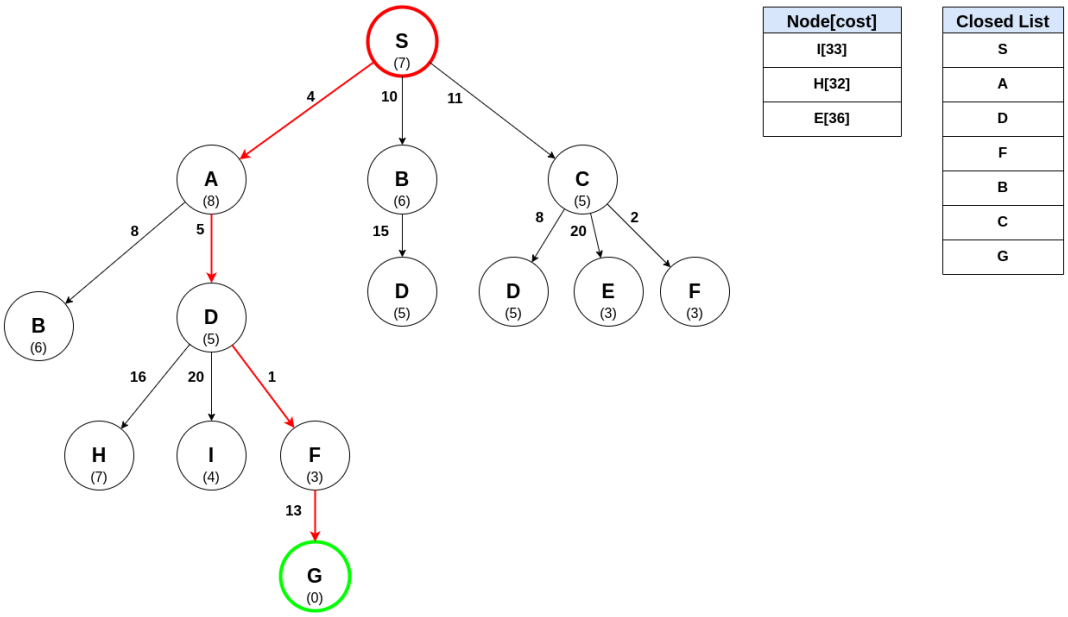
2.1.3 Algoritma A*

A* adalah algoritma pencarian heuristik yang menggabungkan kelebihan UCS dan GBFS. A* menggunakan fungsi evaluasi f(n) = g(n) + h(n), di mana g(n) adalah biaya jalur aktual dari node awal ke node n, dan h(n) adalah perkiraan heuristik dari node n ke tujuan.



Gambar 2.1.3.1. Graf A*

(Sumber : https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search)



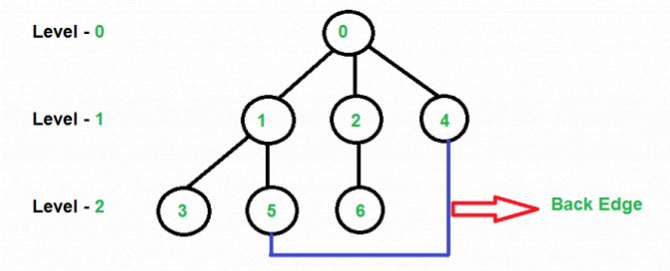
Gambar 2.1.3.2. Pencarian dengan Menggunakan Algoritma A*

(Sumber : https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search)

Algoritma ini menjamin solusi optimal jika heuristiknya konsisten (memenuhi segitiga ketidaksetaraan). A* banyak digunakan dalam bidang seperti game AI, navigasi GPS, dan perencanaan robot, karena efisien dalam menemukan jalur terpendek dengan panduan heuristik yang baik. Kekurangannya adalah memori yang besar diperlukan untuk menyimpan antrian prioritas.

2.1.4 Algoritma Iterative Deepening Depth-First Search (IDDFS)

IDDFS adalah variasi dari Depth-First Search (DFS) yang mengatasi keterbatasan DFS dalam penggunaan memori dan ketidakpastian kedalaman solusi. Algoritma ini menjalankan DFS berulang dengan batas kedalaman yang meningkat secara iteratif (0, 1, 2, dst.) hingga solusi ditemukan.



Gambar 2.1.4.1. Graf IDDFS

(Sumber : https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/)

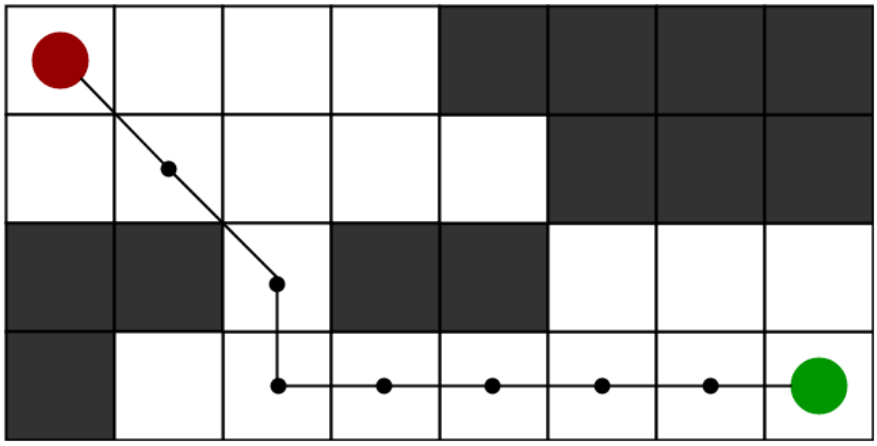
IDDFS berguna untuk masalah dengan ruang pencarian besar, seperti puzzle atau pencarian di graf kompleks, karena menggabungkan keunggulan DFS (memori rendah) dengan sifat lengkap seperti Breadth-First Search (BFS). Namun, IDDFS bisa kurang efisien karena mengulang eksplorasi pada kedalaman sebelumnya.

2.2 Heuristics

2.2.1 Blocking Pieces Heuristic

Heuristik Blocking Pieces merupakan pendekatan estimasi dalam algoritma pencarian berbasis heuristik yang berfokus pada perhitungan jumlah elemen atau objek yang menghalangi jalur langsung menuju solusi optimal. Metode ini tidak mempertimbangkan jarak fisik atau biaya gerakan aktual,

melainkan mengasumsikan bahwa setiap hambatan mewakili langkah tambahan yang diperlukan untuk mencapai tujuan.

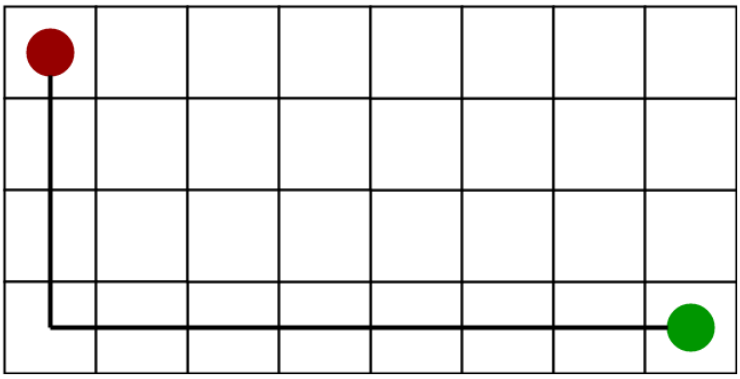


Gambar 2.2.1.1 Gambar Ilustrasi Heuristic Blocking Piece

(Sumber : https://www.geeksforgeeks.org/a-search-algorithm/)

2.2.2 Manhattan Distance Heuristic

Heuristik Manhattan Distance, atau jarak L1, adalah metrik geometris yang mengukur jarak minimum antara dua titik dalam ruang grid melalui gerakan ortogonal (horizontal dan vertikal) dengan rumus $|x_2 - x_1| + |y_2 - y_1|$. Pendekatan ini mengasumsikan bahwa pergerakan hanya dapat terjadi dalam arah sumbu X dan Y, menyerupai pola jalan di kota berbasis grid seperti Manhattan, sehingga sering disebut Taxicab Geometry.



Gambar 2.2.2.1 Ilustrasi Heuristic Manhattan Distance

(Sumber : https://www.geeksforgeeks.org/a-search-algorithm/)

BAB 3 IMPLEMENTASI

3.1 Implementasi Algoritma

Keempat algoritma diimplementasikan dalam JavaScript sebagai bagian dari aplikasi web Rush Hour Solver. Aplikasi ini memungkinkan pengguna untuk memilih algoritma, memuat konfigurasi papan dari file .txt, dan melihat proses pencarian melalui animasi visual serta log teks pencarian. Setiap langkah pencarian dicatat dalam bentuk grid teks (misalnya, PPA., ..A.), dan hasil akhir mencakup informasi seperti jumlah node yang dikunjungi dan waktu eksekusi.

3.2 Alur Proses Pencarian

3.2.1 Algoritma Uniform Cost Search

Berikut adalah alur proses pencarian dengan algoritma Uniform Cost Search :

1. Mulai dengan papan awal dan masukkan ke dalam *priority queue* berdasarkan biaya jalur (jumlah langkah dari awal). Heuristic yang dipilih pada program tidak digunakan pada algoritma ini.
2. Ambil node dengan biaya jalur terendah dari antrian.
3. Periksa apakah papan saat ini adalah solusi.
4. Jika bukan solusi, hasilkan semua langkah yang mungkin, buat papan baru untuk setiap langkah, dan masukkan ke antrian jika belum dikunjungi, dengan biaya jalur bertambah satu.
5. Ulangi langkah 2-4 hingga solusi ditemukan atau antrian kosong.
6. Catat setiap langkah pencarian ke log teks dan perbarui animasi.

3.2.2 Algoritma Greedy Best-First Search

Berikut adalah alur proses pencarian dengan algoritma Greedy Best-First Search :

1. Mulai dengan papan awal dan masukkan ke dalam antrian prioritas berdasarkan nilai heuristik yang dipilih (.
2. Ambil node dengan nilai heuristik terendah dari antrian.
3. Periksa apakah papan saat ini adalah solusi (primary piece mencapai pintu keluar).
4. Jika bukan solusi, hasilkan semua langkah yang mungkin (gerakan mobil), buat papan baru untuk setiap langkah, dan masukkan ke antrian jika belum dikunjungi.
5. Ulangi langkah 2-4 hingga solusi ditemukan atau tidak ada lagi node untuk dieksplorasi.
6. Catat setiap langkah pencarian ke log teks dan perbarui animasi.

3.2.3 Algoritma A*

Berikut adalah alur proses pencarian dengan algoritma A* :

1. Mulai dengan papan awal dan masukkan ke dalam antrian prioritas berdasarkan fungsi evaluasi f(n) = g(n) + h(n), di mana g(n) adalah jumlah langkah dari awal, dan h(n) adalah nilai heuristik.
2. Ambil node dengan nilai f(n) terendah dari antrian.
3. Periksa apakah papan saat ini adalah solusi.
4. Jika bukan solusi, hasilkan semua langkah yang mungkin, buat papan baru untuk setiap langkah, dan masukkan ke antrian jika belum dikunjungi, dengan f(n) yang diperbarui.
5. Ulangi langkah 2-4 hingga solusi ditemukan atau antrian kosong.
6. Catat setiap langkah pencarian ke log teks dan perbarui animasi.

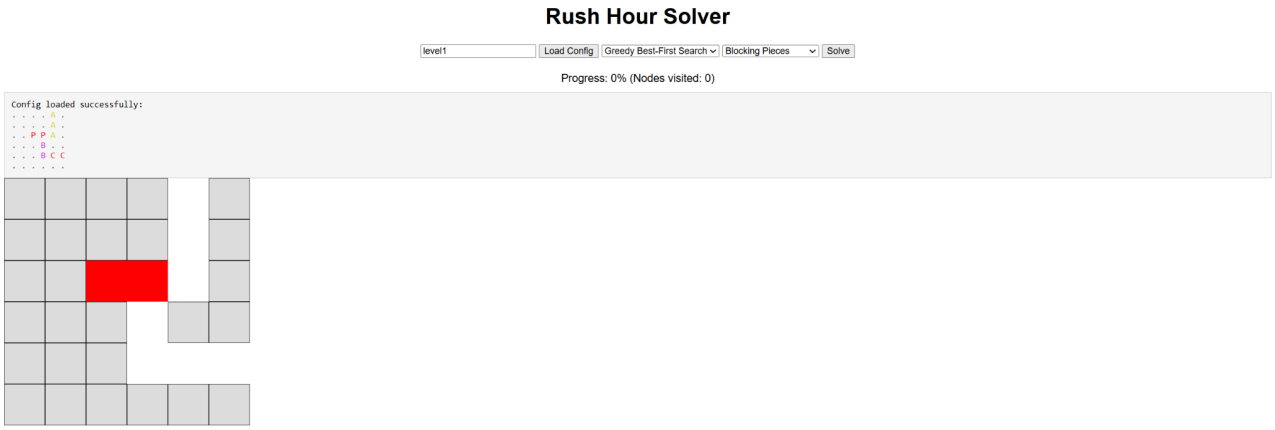
3.2.4 Algoritma Iterative Deepening Depth-First Search (IDDFS)

Berikut adalah alur proses pencarian dengan algoritma Iterative Deepening Depth-First Search :

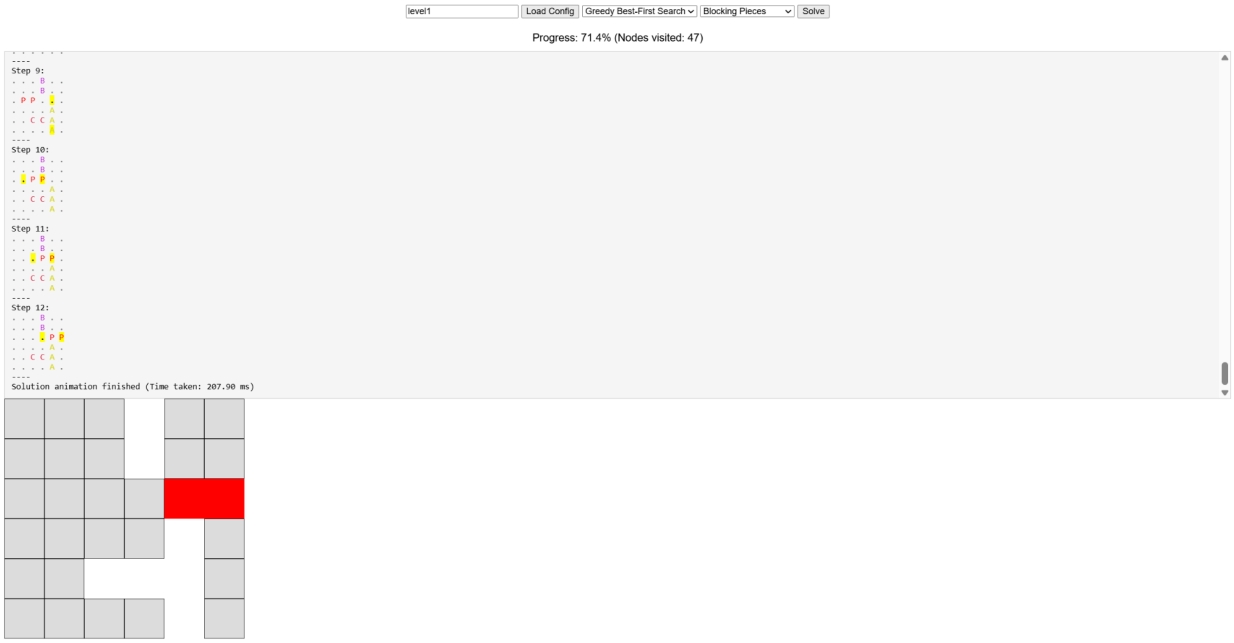
1. Mulai dengan papan awal dan set batas kedalaman awal (depth = 0). Heuristic yang dipilih pada program tidak digunakan pada algoritma ini.
2. Jalankan DFS dengan batasan kedalaman: eksplorasi setiap langkah yang mungkin hingga kedalaman tertentu, dan periksa apakah solusi ditemukan.
3. Jika solusi tidak ditemukan, tingkatkan batas kedalaman (depth++) dan ulangi langkah 2.
4. Untuk setiap langkah dalam DFS, hasilkan papan baru, pastikan belum dikunjungi pada kedalaman tersebut, dan lanjutkan eksplorasi hingga batas kedalaman tercapai.
5. Ulangi hingga solusi ditemukan atau batas maksimum kedalaman tercapai.
6. Catat setiap langkah pencarian ke log teks dan perbarui animasi.

3.3 Tampilan GUI Berbasis Web

Berikut adalah tampilan GUI Berbasis Web sederhana dari program Rush Hour Solver saya.



Gambar 3.3.1 Tampilan GUI Web setelah Load File



Gambar 3.3.2 Tampilan Animasi dan Step by Step penyelesaian

BAB 4 SOURCE CODE

4.1 Struktur Direktori

Program memiliki struktur direktori sebagai berikut:

```
TUCIL3_13523127
|   .gitignore
|   directory_structure.txt
|   index.html
|   LICENSE
|   package-lock.json
|   package.json
|   README.md
|   styles.css
|   temp_structure.txt
|   vite.config.js
|
+---assets
|   \---inputs
|           config.txt
|           input.txt
|           input2.txt
|           level1.txt
|           level2.txt
|           level3.txt
|           level4.txt
|
+---doc
|       dummydoc.txt
|
+---src
|   +---algorithms
|   |       aStar.js
|   |       greedyBFS.js
|   |       heuristics.js
|   |       iddfs.js
|   |       ucs.js
|   |
|   +---core
|   |       board.js
|   |       car.js
|   |       solver.js
|   |
|   +---gui
|   |       interface.js
|   |       sketch.js
|   |
|   \---utils
|           fileReader.js
|           logger.js
|
\---tests
```

4.2 Source Code

4.2.1 index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Rush Hour Solver</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.4.2/p5.min.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        h1 {
            text-align: center;
        }
        .controls {
            text-align: center;
            margin-bottom: 20px;
        }
        #output {
            border: 1px solid #ccc;
            padding: 10px;
            background-color: #f9f9f9;
            max-height: 500px;
            overflow-y: auto;
        }
        pre {
            margin: 0;
            white-space: pre-wrap;
        }
        #progress {
            text-align: center;
            margin-bottom: 10px;
        }
    </style>
</head>
```

```
<body>
    <h1>Rush Hour Solver</h1>
    <div class="controls">
        <input type="text" id="configFileName" placeholder="Enter config file
name (e.g., input)">
        <button onclick="loadConfig()">Load Config</button>
        <select id="algorithm">
            <option value="greedy">Greedy Best-First Search</option>
            <option value="ucs">UCS</option>
            <option value="astar">A*</option>
            <option value="iddfs">Iterative Deepening DFS</option>
        </select>
        <select id="heuristic">
            <option value="blocking">Blocking Pieces</option>
            <option value="manhattan">Manhattan Distance</option>
        </select>
        <button onclick="startSolve()">Solve</button>
    </div>
    <div id="progress">Progress: 0% (Nodes visited: 0)</div>
    <div id="canvas-container"></div>
    <div id="output"></div>
    <script src="src/utils/fileReader.js"></script>
    <script src="src/utils/logger.js"></script>
    <script src="src/core/car.js"></script>
    <script src="src/core/board.js"></script>
    <script src="src/algorithms/heuristics.js"></script>
    <script src="src/algorithms/greedyBFS.js"></script>
    <script src="src/algorithms/ucs.js"></script>
    <script src="src/algorithms/aStar.js"></script>
    <script src="src/algorithms/iddfs.js"></script>
    <script src="src/core/solver.js"></script>
    <script src="src/gui/sketch.js"></script>
    <script src="src/gui/interface.js"></script>
</body>
</html>
```

4.2.2 styles.css

```
body {
    display: flex;
    flex-direction: column;
```

```
    align-items: center;
    font-family: Arial, sans-serif;
}
canvas {
    border: 2px solid black;
    margin: 10px;
}
select, button, input {
    margin: 5px;
    padding: 8px;
    font-size: 16px;
}
#output {
    margin: 10px;
    font-family: monospace;
    white-space: pre;
}
```

4.2.3 vite.config.js

```
import { defineConfig } from 'vite';

export default defineConfig({
  server: {
    port: 3000, // Port untuk development server
    open: true  // Otomatis buka browser saat npm run dev
  },
  base: './' // Pastikan path relatif untuk file statis
});
```

4.2.4 aStar.js

```
function aStar(board, heuristic, callback) {
    const startTime = performance.now();
    let nodesVisited = 0;
    const queue = new PriorityQueue((a, b) => a.cost - b.cost);
    const visited = new Set();
    queue.push({ board, moves: [], cost: heuristic(board), pathCost: 0 });
    visited.add(board.toString());

    while (!queue.isEmpty()) {
```

```
        const { board: currentBoard, moves, pathCost } = queue.pop();
        nodesVisited++;
        if (callback) {
            callback(currentBoard, nodesVisited, moves.length > 0 ?
moves[moves.length - 1].carId : null);
        }
        if (currentBoard.isSolved()) {
            const timeTaken = performance.now() - startTime;
            return { moves, nodesVisited, timeTaken };
        }
        const possibleMoves = currentBoard.generateMoves();
        for (const move of possibleMoves) {
            const newBoard = currentBoard.clone();
            newBoard.applyMove(move);
            const stateStr = newBoard.toString();
            if (!visited.has(stateStr)) {
                visited.add(stateStr);
                const newMoves = [...moves, move];
                const newPathCost = pathCost + 1;
                queue.push({ board: newBoard, moves: newMoves, cost:
newPathCost + heuristic(newBoard), pathCost: newPathCost });
            }
        }
    }
    const timeTaken = performance.now() - startTime;
    return { moves: null, nodesVisited, timeTaken };
}
```

4.2.5 greedyBFS.js

```
function greedyBFS(board, heuristic, callback) {
    const startTime = performance.now();
    let nodesVisited = 0;
    const queue = new PriorityQueue((a, b) => a.cost - b.cost);
    const visited = new Set();
    queue.push({ board, moves: [], cost: heuristic(board) });
    visited.add(board.toString());

    while (!queue.isEmpty()) {
        const { board: currentBoard, moves } = queue.pop();
        nodesVisited++;
        if (callback) {
```

```
            callback(currentBoard, nodesVisited, moves.length > 0 ?
moves[moves.length - 1].carId : null);
        }
        if (currentBoard.isSolved()) {
            const timeTaken = performance.now() - startTime;
            return { moves, nodesVisited, timeTaken };
        }
        const possibleMoves = currentBoard.generateMoves();
        for (const move of possibleMoves) {
            const newBoard = currentBoard.clone();
            newBoard.applyMove(move);
            const stateStr = newBoard.toString();
            if (!visited.has(stateStr)) {
                visited.add(stateStr);
                const newMoves = [...moves, move];
                queue.push({ board: newBoard, moves: newMoves, cost:
heuristic(newBoard) });
            }
        }
    }
    const timeTaken = performance.now() - startTime;
    return { moves: null, nodesVisited, timeTaken };
}
```

4.2.6 heuristics.js

```
function blockingPieces(board) {
    const primaryCar = board.cars.find(car => car.id === 'P');
    if (!primaryCar.isHorizontal || primaryCar.row !== board.exit.row) return
Infinity;
    let count = 0;
    for (let col = primaryCar.col + primaryCar.length; col < board.exit.col;
col++) {
        if (board.grid[primaryCar.row][col] !== 0) count++;
    }
    return count;
}

function manhattanDistance(board) {
    const primaryCar = board.cars.find(car => car.id === 'P');
    if (!primaryCar.isHorizontal || primaryCar.row !== board.exit.row) return
Infinity;
```

```
    return Math.abs((primaryCar.col + primaryCar.length - 1) -
board.exit.col);
}
```

4.2.7 iddfs.js

```
function iddfs(board, unused, callback) {
    const startTime = performance.now();
    let nodesVisited = 0;
    const visitedAtDepth = new Map();

    function depthLimitedDFS(currentBoard, depth, moves, visited) {
        nodesVisited++;
        console.log('IDDFS - Visiting node:', nodesVisited, 'Depth:', depth,
'State:', currentBoard.toString());
        if (callback) {
            callback(currentBoard, nodesVisited, moves.length > 0 ?
moves[moves.length - 1].carId : null);
        }
        if (currentBoard.isSolved()) {
            return { moves, found: true };
        }
        if (depth <= 0) {
            return { moves: null, found: false };
        }

        const possibleMoves = currentBoard.generateMoves();
        for (const move of possibleMoves) {
            const newBoard = currentBoard.clone();
            newBoard.applyMove(move);
            const stateStr = newBoard.toString();
            const prevDepth = visitedAtDepth.get(stateStr) || -1;
            if (prevDepth < depth) {
                visitedAtDepth.set(stateStr, depth);
                const newMoves = [...moves, move];
                const result = depthLimitedDFS(newBoard, depth - 1, newMoves,
visited);
                if (result.found) {
                    return result;
                }
            }
        }
```

```
        return { moves: null, found: false };
    }

    let depth = 0;
    let result = { moves: null, found: false };
    while (!result.found) {
        console.log('IDDFS - Starting depth:', depth);
        visitedAtDepth.clear();
        visitedAtDepth.set(board.toString(), depth);
        result = depthLimitedDFS(board.clone(), depth, [], new Set());
        depth++;
        if (depth > 1000) { // Batas untuk mencegah loop tak terbatas
            break;
        }
    }

    const timeTaken = performance.now() - startTime;
    if (result.found) {
        console.log('IDDFS - Solution found, total nodes visited:',
nodesVisited);
        return { moves: result.moves, nodesVisited, timeTaken };
    } else {
        console.log('IDDFS - No solution found, total nodes visited:',
nodesVisited);
        return { moves: null, nodesVisited, timeTaken };
    }
}
```

4.2.8 ucs.js

```
function ucs(board, unused, callback) {
    const startTime = performance.now();
    let nodesVisited = 0;
    const queue = new PriorityQueue((a, b) => a.cost - b.cost);
    const visited = new Set();
    queue.push({ board: board.clone(), moves: [], cost: 0 });
    visited.add(board.toString());

    while (!queue.isEmpty()) {
        const { board: currentBoard, moves, cost } = queue.pop();
        nodesVisited++;
        console.log('UCS - Visiting node:', nodesVisited, 'State:',
```

```
currentBoard.toString(), 'Cost:', cost);
        if (callback) {
            callback(currentBoard, nodesVisited, moves.length > 0 ?
moves[moves.length - 1].carId : null);
        }
        if (currentBoard.isSolved()) {
            const timeTaken = performance.now() - startTime;
            console.log('UCS - Solution found, total nodes visited:',
nodesVisited);
            return { moves, nodesVisited, timeTaken };
        }
        const possibleMoves = currentBoard.generateMoves();
        for (const move of possibleMoves) {
            const newBoard = currentBoard.clone();
            newBoard.applyMove(move);
            const stateStr = newBoard.toString();
            if (!visited.has(stateStr)) {
                visited.add(stateStr);
                const newMoves = [...moves, move];
                queue.push({ board: newBoard, moves: newMoves, cost: cost + 1
});
            }
        }
    }
    const timeTaken = performance.now() - startTime;
    console.log('UCS - No solution found, total nodes visited:',
nodesVisited);
    return { moves: null, nodesVisited, timeTaken };
}
```

4.2.9 board.js

```
class Board {
    constructor() {
        this.width = 0;
        this.height = 0;
        this.grid = [];
        this.cars = [];
        this.exit = { row: -1, col: -1 }; // Posisi K di luar papan
    }

    // Parse input .txt
```

```
    parseConfig(content) {
        console.log('Parsing content:', content); // Debug: Cetak konten yang
diterima
        const lines = content.trim().split('\n');
        // Baris pertama: A B (dimensi)
        const [width, height] = lines[0].split(' ').map(Number);
        this.width = width;
        this.height = height;
        // Baris kedua: N (jumlah balok)
        const numCars = parseInt(lines[1]);
        // Baris berikutnya: konfigurasi papan
        const gridLines = lines.slice(2, 2 + height);
        this.grid = gridLines.map(line => line.trim().split('').map(c => c ===
'.' ? 0 : c));

        console.log('Parsed grid:', this.grid); // Debug: Cetak grid setelah
parsing

        // Identifikasi balok
        const carMap = new Map(); // Map untuk melacak balok berdasarkan ID
        for (let row = 0; row < height; row++) {
            for (let col = 0; col < width; col++) {
                const id = this.grid[row][col];
                if (id === 0) continue; // Sel kosong
                if (!carMap.has(id)) {
                    carMap.set(id, { id, positions: [] });
                }
                carMap.get(id).positions.push({ row, col });
            }
        }

        // Konversi posisi ke objek Car
        this.cars = Array.from(carMap.values()).map(carData => {
            const { id, positions } = carData;
            positions.sort((a, b) => a.row === b.row ? a.col - b.col : a.row -
b.row);
            const isHorizontal = positions[0].row === positions[1]?.row ||
false;
            const length = positions.length;
            const row = positions[0].row;
            const col = positions[0].col;
            const color = id === 'P' ? 'red' : `hsl(${Math.random() * 360},
70%, 50%)`;
```

```
            return new Car(id, row, col, length, isHorizontal, color);
        });

        // Tentukan posisi exit (K) berdasarkan primary piece (P)
        const primaryCar = this.cars.find(car => car.id === 'P');
        if (!primaryCar) {
            throw new Error('Primary piece (P) tidak ditemukan');
        }

        // Tentukan posisi K di luar papan, sejajar dengan P
        if (primaryCar.isHorizontal) {
            this.exit.col = this.width; // K di dinding kanan, di luar papan
            this.exit.row = primaryCar.row;
        } else {
            this.exit.row = this.height; // K di dinding bawah, di luar papan
            this.exit.col = primaryCar.col;
        }
        console.log(`Exit set at (${this.exit.row}, ${this.exit.col}) outside
board`); // Debug

        // Validasi
        if (this.cars.length !== numCars + 1) { // +1 untuk primary piece
            console.warn('Number of cars mismatch:', this.cars.length,
'expected', numCars + 1);
            throw new Error('Jumlah balok tidak sesuai dengan N');
        }
    }

    // Salin papan untuk state baru
    clone() {
        const newBoard = new Board();
        newBoard.width = this.width;
        newBoard.height = this.height;
        newBoard.grid = this.grid.map(row => [...row]);
        newBoard.cars = this.cars.map(car => car.clone());
        newBoard.exit = { ...this.exit };
        return newBoard;
    }

    // Terapkan gerakan
    applyMove(move) {
        const car = this.cars.find(c => c.id === move.carId);
        if (!car) return false;
```

```
        if (car.isHorizontal) {
            for (let i = 0; i < car.length; i++) {
                this.grid[car.row][car.col + i] = 0;
            }
            car.col += move.direction;
            for (let i = 0; i < car.length; i++) {
                this.grid[car.row][car.col + i] = car.id;
            }
        } else {
            for (let i = 0; i < car.length; i++) {
                this.grid[car.row + i][car.col] = 0;
            }
            car.row += move.direction;
            for (let i = 0; i < car.length; i++) {
                this.grid[car.row + i][car.col] = car.id;
            }
        }
        return true;
    }

    // Generate kemungkinan gerakan
    generateMoves() {
        const moves = [];
        for (const car of this.cars) {
            if (car.isHorizontal) {
                if (car.col > 0 && this.grid[car.row][car.col - 1] === 0) {
                    moves.push({ carId: car.id, direction: -1 });
                }
                if (car.col + car.length < this.width &&
this.grid[car.row][car.col + car.length] === 0) {
                    moves.push({ carId: car.id, direction: 1 });
                }
            } else {
                if (car.row > 0 && this.grid[car.row - 1][car.col] === 0) {
                    moves.push({ carId: car.id, direction: -1 });
                }
                if (car.row + car.length < this.height && this.grid[car.row +
car.length][car.col] === 0) {
                    moves.push({ carId: car.id, direction: 1 });
                }
            }
        }
        return moves;
```

```
    }

    // Cek apakah selesai
    isSolved() {
        const primaryCar = this.cars.find(car => car.id === 'P');
        return primaryCar.isHorizontal && primaryCar.col + primaryCar.length
=== this.exit.col && primaryCar.row === this.exit.row ||
               !primaryCar.isHorizontal && primaryCar.row + primaryCar.length
=== this.exit.row && primaryCar.col === this.exit.col;
    }

    // Konversi ke string untuk visited set
    toString() {
        return this.cars.map(car =>
`${car.id}:${car.row},${car.col}`).join('|');
    }
}
```

4.2.10 car.js

```
class Car {
    constructor(id, row, col, length, isHorizontal, color) {
        this.id = id;
        this.row = row;
        this.col = col;
        this.length = length;
        this.isHorizontal = isHorizontal;
        this.color = color;
    }

    clone() {
        return new Car(this.id, this.row, this.col, this.length,
this.isHorizontal, this.color);
    }
}
```

4.2.11 solver.js

```
class Solver {
    constructor() {
        this.board = null;
```

```
    }

    loadConfig(content) {
        this.board = new Board();
        this.board.parseConfig(content);
    }

    solve(algorithm, heuristicName) {
        if (!this.board) throw new Error('No board loaded');
        const heuristic = heuristicName === 'blocking' ? blockingPieces :
manhattanDistance;
        let result;
        switch (algorithm) {
            case 'greedy':
                result = greedyBFS(this.board, heuristic);
                break;
            case 'ucs':
                result = ucs(this.board);
                break;
            case 'astar':
                result = aStar(this.board, heuristic);
                break;
            default:
                throw new Error('Invalid algorithm');
        }
        return result;
    }
}
```

4.2.12 interface.js

```
class PriorityQueue {
    constructor(comparator) {
        this.items = [];
        this.comparator = comparator;
    }

    push(item) {
        this.items.push(item);
        this.items.sort(this.comparator);
    }
```

```
    pop() {
        return this.items.shift();
    }

    isEmpty() {
        return this.items.length === 0;
    }
}

let solver = new Solver();
window.solver = solver;
window.previousBoard = null;

function loadConfig() {
    const fileName = document.getElementById('configFileName').value.trim();
    if (!fileName) {
        alert('Please enter a config file name');
        return;
    }
    window.configFileName = fileName; // Simpan nama file konfigurasi untuk
digunakan saat menyimpan solusi
    readConfigFile(fileName, (content) => {
        try {
            solver.loadConfig(content);
            window.previousBoard = solver.board.clone();
            const primaryCar = solver.board.cars.find(c => c.id === 'P');
            if (primaryCar && solver.board.exit) {
                if (primaryCar.isHorizontal && solver.board.exit.row !==
primaryCar.row) {
                    throw new Error('Exit (K) must be on the same row as
horizontal primary car (P)');
                } else if (!primaryCar.isHorizontal && solver.board.exit.col
!== primaryCar.col) {
                    throw new Error('Exit (K) must be on the same column as
vertical primary car (P)');
                }
            }
            document.getElementById('output').innerHTML = '<pre>Config loaded
successfully:\n' + logBoard(solver.board) + '</pre>';
            if (typeof window.updateCanvasSize === 'function') {
                window.updateCanvasSize(solver.board.width,
solver.board.height);
            } else {
```

```
                console.warn('updateCanvasSize is not defined.');
            }
        } catch (e) {
            alert('Error parsing config: ' + e.message);
            document.getElementById('output').innerHTML = '<pre>Error: ' +
e.message + '</pre>';
        }
    });
}

function startSolve() {
    if (!solver.board) {
        alert('Please load a config file first');
        return;
    }
    window.solutionSteps = []; // Reset langkah-langkah sebelum memulai
pencarian
    window.searchAlgorithm = null;
    window.searchHeuristic = null;
    window.searchStep = 0;
    window.searchResult = null;
    window.isSearching = false;
    window.nodesVisited = 0;
    window.currentBoard = null;
    window.previousBoard = solver.board.clone();
    document.getElementById('progress').textContent = 'Progress: 0% (Nodes
visited: 0)';
    document.getElementById('output').innerHTML = '<pre>Starting
search...</pre>';

    const algorithm = document.getElementById('algorithm').value;
    const heuristic = document.getElementById('heuristic').value;
    try {
        window.searchAlgorithm = algorithm;
        window.searchHeuristic = heuristic;
        window.searchStep = 0;
        window.searchResult = null;
        window.isSearching = true;
        window.nodesVisited = 0;
        window.currentBoard = solver.board.clone();
        console.log('startSolve called with algorithm:', algorithm,
'heuristic:', heuristic);
        console.log('Global variables after reset:', {
```

```
            searchAlgorithm: window.searchAlgorithm,
            searchHeuristic: window.searchHeuristic,
            isSearching: window.isSearching,
            nodesVisited: window.nodesVisited
        });
    } catch (e) {
        alert('Error starting solve: ' + e.message);
        document.getElementById('output').innerHTML = '<pre>Error: ' +
e.message + '</pre>';
        console.error('Error in startSolve:', e);
    }
}
```

4.2.13 sketch.js

```
let solution = null;
let currentStep = 0;
let isSolving = false;
const CELL_SIZE = 60;
let estimatedTotalNodes = 1000;
let solutionSteps = []; // Array untuk menyimpan langkah-langkah
let configFileName = ''; // Untuk menyimpan nama file konfigurasi

// Fungsi untuk menghitung jarak primary piece ke pintu keluar
function calculateDistanceToExit(board) {
    const primaryCar = board?.cars?.find(car => car.id === 'P');
    if (!primaryCar || !board.exit) return 0;
    let distance;
    if (primaryCar.isHorizontal) {
        distance = Math.abs((primaryCar.col + primaryCar.length) -
(board.exit.col >= 0 ? board.exit.col : board.width));
    } else {
        distance = Math.abs((primaryCar.row + primaryCar.length) -
(board.exit.row >= 0 ? board.exit.row : board.height));
    }
    return distance;
}

// Fungsi untuk mengubah ukuran kanvas
window.updateCanvasSize = function(width, height) {
    console.log('Updating canvas size to:', width, height);
    if (typeof resizeCanvas === 'function') {
```

```
        resizeCanvas(width * CELL_SIZE, height * CELL_SIZE);
    } else {
        console.error('p5.js not ready');
    }
};

// Fungsi untuk menyimpan langkah-langkah ke file .txt
function saveSolutionToFile(steps, configName) {
    const fileName = configName.replace(/\.[^/.]+$/, "") + 'solution.txt'; //
Hapus ekstensi dan tambahkan "solution.txt"
    const content = steps.join('\n');
    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = fileName;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    URL.revokeObjectURL(url);
    console.log(`Solution saved as ${fileName}`);
}

function setup() {
    if (window.solver && window.solver.board) {
        createCanvas(window.solver.board.width * CELL_SIZE,
window.solver.board.height * CELL_SIZE);
        estimatedTotalNodes = window.solver.board.width *
window.solver.board.height * 50;
    } else {
        createCanvas(360, 360);
    }
    frameRate(60);
    console.log('Setup completed, solver available:', !!window.solver);
}

function draw() {
    if (!window.solver || !window.solver.board) {
        console.log('Draw skipped: solver or board not available');
        return;
    }

    background(220);
```

```
    // Gambar grid
    for (let i = 0; i < window.solver.board.height; i++) {
        for (let j = 0; j < window.solver.board.width; j++) {
            stroke(0);
            noFill();
            rect(j * CELL_SIZE, i * CELL_SIZE, CELL_SIZE, CELL_SIZE);
        }
    }

    // Gunakan currentBoard untuk animasi pencarian
    const boardToDraw = window.currentBoard || window.solver.board;
    console.log('boardToDraw:', boardToDraw);

    // Gambar balok-balok berwarna
    if (boardToDraw && Array.isArray(boardToDraw.cars)) {
        for (let car of boardToDraw.cars) {
            fill(car.color);
            noStroke();
            if (car.isHorizontal) {
                rect(car.col * CELL_SIZE, car.row * CELL_SIZE, car.length *
CELL_SIZE, CELL_SIZE);
            } else {
                rect(car.col * CELL_SIZE, car.row * CELL_SIZE, CELL_SIZE,
car.length * CELL_SIZE);
            }
        }
    } else {
        console.error('boardToDraw.cars is not iterable:', boardToDraw?.cars);
    }

    // Gambar kotak exit di luar papan
    if (boardToDraw?.exit) {
        fill('blue');
        noStroke();
        if (boardToDraw.exit.col === -1) {
            rect(-CELL_SIZE, boardToDraw.exit.row * CELL_SIZE, CELL_SIZE,
CELL_SIZE);
        } else if (boardToDraw.exit.col === boardToDraw.width) {
            rect(boardToDraw.width * CELL_SIZE, boardToDraw.exit.row *
CELL_SIZE, CELL_SIZE, CELL_SIZE);
        } else if (boardToDraw.exit.row === -1) {
            rect(boardToDraw.exit.col * CELL_SIZE, -CELL_SIZE, CELL_SIZE,
```

```
CELL_SIZE);
        } else if (boardToDraw.exit.row === boardToDraw.height) {
            rect(boardToDraw.exit.col * CELL_SIZE, boardToDraw.height *
CELL_SIZE, CELL_SIZE, CELL_SIZE);
        }
    }

    // Animasi pencarian
    if (window.isSearching && window.searchAlgorithm) {
        console.log('Starting search loop, algorithm:',
window.searchAlgorithm, 'nodesVisited:', window.nodesVisited);
        let output =
document.getElementById('output').innerHTML.replace('</pre>', '');
        let stepOutput = [];
        if (window.previousBoard) {
            const log = logBoardWithChanges(window.previousBoard,
boardToDraw);
            output += log;
            stepOutput.push(log);
        } else {
            const log = logBoard(boardToDraw);
            output += log;
            stepOutput.push(log);
        }
        output += '----\n';
        stepOutput.push('----');
        output += '</pre>';
        document.getElementById('output').innerHTML = output;
        document.getElementById('output').scrollTop =
document.getElementById('output').scrollHeight;
        solutionSteps.push(...stepOutput);

        if (!window.searchResult) {
            let heuristicFunc = window.searchHeuristic === 'blocking' ?
blockingPieces : manhattanDistance;
            let algorithmFunc;
            switch (window.searchAlgorithm) {
                case 'greedy':
                    algorithmFunc = greedyBFS;
                    break;
                case 'ucs':
                    algorithmFunc = ucs;
                    break;
```

```
                case 'astar':
                    algorithmFunc = aStar;
                    break;
                case 'iddfs':
                    algorithmFunc = iddfs;
                    break;
                default:
                    console.error('Unknown algorithm:',
window.searchAlgorithm);
                    window.isSearching = false;
                    document.getElementById('output').innerHTML = '<pre>Error:
Unknown algorithm</pre>';
                    return;
            }
            console.log('Executing algorithm:', window.searchAlgorithm);
            try {
                window.searchResult = algorithmFunc(window.currentBoard ||
window.solver.board.clone(), window.searchAlgorithm !== 'ucs' &&
window.searchAlgorithm !== 'iddfs' ? heuristicFunc : null, (board, visited,
carId) => {
                    console.log('Callback triggered - Received nodesVisited:',
visited, 'Car ID:', carId);
                    window.previousBoard = window.currentBoard ?
window.currentBoard.clone() : window.solver.board.clone();
                    window.currentBoard = board;
                    window.nodesVisited = visited;
                    console.log('Updated window.nodesVisited to:',
window.nodesVisited);
                    if (window.nodesVisited > estimatedTotalNodes) {
                        estimatedTotalNodes = window.nodesVisited * 2;
                    }
                    updateProgress(board);
                    let output =
document.getElementById('output').innerHTML.replace('</pre>', '');
                    let stepOutput = [];
                    if (carId) {
                        const moveLog = `Moved car ${carId}:`;
                        output += moveLog + '\n';
                        stepOutput.push(moveLog);
                    }
                    const boardLog = logBoard(board);
                    output += boardLog;
                    stepOutput.push(boardLog);
```

```
                    output += '----\n';
                    stepOutput.push('----');
                    output += '</pre>';
                    document.getElementById('output').innerHTML = output;
                    document.getElementById('output').scrollTop =
document.getElementById('output').scrollHeight;
                    solutionSteps.push(...stepOutput);
                });
                if (window.searchResult) {
                    console.log('Search completed, result:',
window.searchResult);
                    window.nodesVisited = window.searchResult.nodesVisited;
                    console.log('Final window.nodesVisited set to:',
window.nodesVisited);
                    let finalOutput =
document.getElementById('output').innerHTML.replace('</pre>', '');
                    let finalSteps = [];
                    if (window.searchResult.moves) {
                        const boardLog = logBoard(window.currentBoard);
                        finalOutput += boardLog;
                        finalSteps.push(boardLog);
                        finalOutput += '----\n';
                        finalSteps.push('----');
                        const solutionFound = 'Solution found: ' +
window.searchResult.moves.length + ' moves';
                        finalOutput += solutionFound + '\n';
                        finalSteps.push(solutionFound);
                        const nodesVisited = 'Total nodes visited: ' +
window.nodesVisited;
                        finalOutput += nodesVisited + '\n';
                        finalSteps.push(nodesVisited);
                        const timeTaken = 'Time taken: ' +
window.searchResult.timeTaken.toFixed(2) + ' ms';
                        finalOutput += timeTaken + '\n';
                        finalSteps.push(timeTaken);
                        document.getElementById('output').innerHTML =
finalOutput + '</pre>';
                        solutionSteps.push(...finalSteps);
                        solution = window.searchResult.moves;
                        window.isSearching = false;
                        isSolving = true;
                        currentStep = 0;
                        updateProgress(window.currentBoard);
```

```
                    } else {
                        const noSolution = 'No solution found';
                        finalOutput += noSolution + '\n';
                        finalSteps.push(noSolution);
                        finalOutput += 'Total nodes visited: ' +
window.nodesVisited + '\n';
                        finalSteps.push('Total nodes visited: ' +
window.nodesVisited);
                        finalOutput += 'Time taken: ' +
window.searchResult.timeTaken.toFixed(2) + ' ms\n';
                        finalSteps.push('Time taken: ' +
window.searchResult.timeTaken.toFixed(2) + ' ms');
                        document.getElementById('output').innerHTML =
finalOutput + '</pre>';
                        solutionSteps.push(...finalSteps);
                        window.isSearching = false;
                        alert('No solution found');
                    }
                    // Simpan langkah-langkah ke file setelah pencarian
selesai
                    saveSolutionToFile(solutionSteps, configFileName);
                }
            } catch (e) {
                document.getElementById('output').innerHTML = '<pre>Error
running algorithm: ' + e.message + '</pre>';
                console.error('Error running algorithm:', e);
                window.isSearching = false;
                alert('Error during search: ' + e.message);
            }
        }
    }

    // Animasi solusi
    if (isSolving && solution && currentStep < solution.length) {
        if (frameCount % 30 === 0) {
            window.solver.board.applyMove(solution[currentStep]);
            let output =
document.getElementById('output').innerHTML.replace('</pre>', '');
            let stepOutput = [];
            const stepLog = `Step ${currentStep + 1}:`;
            output += stepLog + '\n';
            stepOutput.push(stepLog);
            if (window.previousBoard) {
```

```
                const log = logBoardWithChanges(window.previousBoard,
window.solver.board);
                output += log;
                stepOutput.push(log);
            } else {
                const log = logBoard(window.solver.board);
                output += log;
                stepOutput.push(log);
            }
            output += '----\n';
            stepOutput.push('----');
            window.previousBoard = window.solver.board.clone();
            window.currentBoard = window.solver.board;
            currentStep++;
            if (currentStep >= solution.length) {
                isSolving = false;
                const finishedLog = 'Solution animation finished (Time taken:
' + window.searchResult.timeTaken.toFixed(2) + ' ms)';
                output += finishedLog + '\n';
                stepOutput.push(finishedLog);
            }
            output += '</pre>';
            document.getElementById('output').innerHTML = output;
            document.getElementById('output').scrollTop =
document.getElementById('output').scrollHeight;
            solutionSteps.push(...stepOutput);
            // Simpan ulang setelah animasi selesai
            if (!isSolving) {
                saveSolutionToFile(solutionSteps, configFileName);
            }
        }
    }
}

// Fungsi untuk memperbarui kemajuan
function updateProgress(board) {
    const progressElement = document.getElementById('progress');
    if (progressElement && window.nodesVisited >= 0) {
        const initialBoard = window.solver.board;
        const initialDistance = calculateDistanceToExit(initialBoard);
        const currentDistance = calculateDistanceToExit(board);
        let distanceProgress = initialDistance > 0 ? ((initialDistance -
currentDistance) / initialDistance) * 100 : 100;
```

```
        let nodeProgress = (window.nodesVisited / estimatedTotalNodes) * 100;
        const combinedProgress = Math.min((0.7 * distanceProgress + 0.3 *
nodeProgress), 100);
        progressElement.textContent = `Progress:
${combinedProgress.toFixed(1)}% (Nodes visited: ${window.nodesVisited})`;
        console.log('Progress updated:', { distanceProgress, nodeProgress,
combinedProgress, nodesVisited: window.nodesVisited });
    }
}
```

4.2.14 fileReader.js

```
// Membaca file .txt dari assets/inputs/ berdasarkan nama file
function readConfigFile(fileName, callback) {
    const filePath = `assets/inputs/${fileName}.txt`;
    fetch(filePath)
        .then(response => {
            if (!response.ok) {
                throw new Error(`File ${fileName}.txt not found in
assets/inputs/`);
            }
            return response.text();
        })
        .then(content => {
            console.log('File content:', content); // Debug: Cetak konten file
            callback(content);
        })
        .catch(error => {
            alert('Error reading config file: ' + error.message);
            console.error('Fetch error:', error);
        });
}
```

4.2.15 logger.js

```
function logBoard(board) {
    let output = '';
    for (let i = 0; i < board.height; i++) {
        for (let j = 0; j < board.width; j++) {
            const cell = board.grid[i][j];
            if (cell === 0) {
```

```
                output += '<span style="color: grey;">.</span> ';
            } else {
                // Cari mobil untuk menentukan warnanya
                const car = board.cars.find(c => c.id === cell);
                const color = car ? car.color : 'black';
                output += `<span style="color: ${color};">${cell}</span> `;
            }
        }
        output += '\n';
    }
    return output;
}

// Fungsi untuk membandingkan dua papan dan menyorot perubahan
function logBoardWithChanges(oldBoard, newBoard) {
    let output = '';
    for (let i = 0; i < newBoard.height; i++) {
        for (let j = 0; j < newBoard.width; j++) {
            const oldCell = oldBoard.grid[i][j];
            const newCell = newBoard.grid[i][j];
            if (oldCell !== newCell) {
                // Jika ada perubahan, sorot dengan latar belakang kuning
                const car = newBoard.cars.find(c => c.id === newCell);
                const color = car ? car.color : 'black';
                output += `<span style="color: ${color}; background-color:
yellow;">${newCell === 0 ? '.' : newCell}</span> `;
            } else {
                const cell = newBoard.grid[i][j];
                if (cell === 0) {
                    output += '<span style="color: grey;">.</span> ';
                } else {
                    const car = newBoard.cars.find(c => c.id === cell);
                    const color = car ? car.color : 'black';
                    output += `<span style="color: ${color};">${cell}</span>
`;
                }
            }
        }
        output += '\n';
    }
    return output;
}
```

4.3 Class dan Method

Class yang digunakan

- **Solver**

  Kelas utama yang mengelola papan permainan dan proses penyelesaian. Ini adalah inti logika permainan Rush Hour. Fungsi utamanya adalah sebagai berikut :

  - loadConfig(content): Memuat konfigurasi papan dari file teks.
  - board: Properti yang menyimpan objek papan permainan saat ini.

- **Board**

  Sebagai papan permainan Rush Hour, termasuk posisi mobil dan aturan gerakan. Fungsi utamanya adalah sebagai berikut :

  clone(): Membuat salinan papan untuk simulasi langkah.

  - generateMoves(): Menghasilkan semua gerakan yang mungkin dari papan saat ini.
  - applyMove(move): Menerapkan gerakan tertentu ke papan.
  - isSolved(): Memeriksa apakah papan telah mencapai solusi (primary piece keluar).
  - toString(): Mengonversi papan ke string untuk pelacakan status.

- **Car**

  Mewakili mobil di papan, dengan atribut seperti posisi, orientasi, dan panjang.

  - id: Identifikasi unik mobil (misalnya, 'P' untuk primary car).
  - col, row: Posisi awal mobil.
  - length: Panjang mobil.
  - isHorizontal: Menunjukkan apakah mobil berorientasi horizontal.

- **PriorityQueue**

  Struktur data untuk mengelola antrian prioritas, digunakan oleh algoritma pencarian seperti GBFS, UCS, dan A*.

  - push(item): Menambahkan item ke antrian berdasarkan prioritas.
  - pop(): Mengambil item dengan prioritas tertinggi.
  - isEmpty(): Memeriksa apakah antrian kosong.

Method-method utama

1. **Algoritma Pencarian (di File Berbeda)**
    - **greedyBFS(board, heuristic, callback)**: Mencari solusi menggunakan GBFS dengan prioritas berdasarkan heuristik.
    - **ucs(board, unused, callback)**: Mencari solusi menggunakan UCS berdasarkan biaya jalur minimum.
    - **aStar(board, heuristic, callback)**: Mencari solusi menggunakan A* dengan kombinasi biaya jalur dan heuristik.
    - **iddfs(board, unused, callback)**: Mencari solusi menggunakan IDDFS dengan batasan kedalaman iteratif.
    - **Fungsi Umum**: Semua algoritma menggunakan callback untuk melacak langkah pencarian, menghitung nodesVisited, dan mengembalikan moves, nodesVisited, serta timeTaken.
2. **Heuristik (di heuristics.js)**
    - **blockingPieces(board)**: Menghitung jumlah mobil yang menghalangi primary car sebagai nilai heuristik.
    - **manhattanDistance(board)**: Menghitung jarak Manhattan dari primary car ke pintu keluar sebagai heuristik.
3. **Visualisasi dan UI (di sketch.js dan interface.js)**
    - **setup()**: Menginisialisasi kanvas p5.js untuk animasi berdasarkan ukuran papan.
    - **draw()**: Menggambar papan dan animasi gerakan mobil, serta mengelola log pencarian.
    - **updateProgress(board)**: Memperbarui elemen progress dengan persentase dan jumlah node yang dikunjungi.
    - **loadConfig()**: Memuat konfigurasi papan dari input pengguna.
    - **startSolve()**: Memulai proses pencarian dengan algoritma yang dipilih.
4. **Logger (di logger.js)**
    - **logBoard(board)**: Mengonversi papan ke format teks grid (misalnya, PPA., ..A.).
    - **logBoardWithChanges(prevBoard, currentBoard)**: Menampilkan perubahan papan dalam bentuk grid.

BAB 5 PENGUJIAN DAN ANALISIS

5.1 Pengujian

Program akan menampilkan log node-node yang ia kunjungi di atas hasil step by step gerakan penyelesaian yang ditemukan. Namun pada laporan saya tidak akan menunjukkan log node-node karena dirasa kurang efisien. Terdapat 4 test case yang akan digunakan untuk masing-masing algoritma, yang diambil dari game rush hour online level 1 hingga level 4.

Berikut adalah configurasi test case yang digunakan :

1. 6 6
   3
   ....A.
   ....A.
   ..PPA.K
   ...B..
   ...BCC
   ......
2. 6 6
   5
   A....B
   A....B
   PP...BK
  CC
   ..E...
   ..EDDD
3. 6 6
   5
   ..A...
   ..A..E
   PPA..EK
   .B..D.
   .BCCD.
   ....D.
4. 6 6
   5

..BBBE.

..A..E

PPA..EK

.....

..CCDD

.....

5.1.1 Algoritma Greedy Best-First Search

5.1.1.1 Test Case 1 “level1.txt” Heuristic Manhattan Distance

Solution found: 12 moves
Total nodes visited: 28
Time taken: 107.60 ms
Step 1:
. . . . A .
. . . . A .
. P P . A .
. . . B . .
. . . B C C
. . . . . .

Step 2:
. . . . A .
. . . . A .
. P P B A .
. . . B . .
. . . . C C
. . . . . .

Step 3:
. . . . A .
. . . B A .
. P P B A .
. . . . . .
. . . . C C
. . . . . .

Step 4:
. . . B A .
. . . B A .
. P P . A .

. . . . . .
. . . . C C
. . . . . .

Step 5:
. . . B A .
. . . B A .
. . P P A .
. . . . . .
. . . . C C
. . . . . .

Step 6:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . . . C C
. . . . . .

Step 7:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . . C C .
. . . . . .

Step 8:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . C C . .
. . . . . .

Step 9:
. . . B . .
. . . B . .
. . P P A .

. . . . A .
. . C C A .
. . . . . .

Step 10:
. . . B . .
. . . B . .
. . P P . .
. . . . A .
. . C C A .
. . . . A .

Step 11:
. . . B . .
. . . B . .
. . . P P .
. . . . A .
. . C C A .
. . . . A .

Step 12:
. . . B . .
. . . B . .
. . . . P P
. . . . A .
. . C C A .
. . . . A .

Solution animation finished (Time taken: 107.60 ms)

5.1.1.2 Test Case 2 “level2.txt” Heuristic Manhattan Distance

Solution found: 10 moves
Total nodes visited: 154
Time taken: 3242.80 ms
Step 1:
A . . . . B
A . . . . B
. P P . . . B
. . . . C C
. . E . . .

. . E D D D

Step 2:
A . . . . B
A . . . . B
. . P P . B
. . . . C C
. . E . . .
. . E D D D

Step 3:
A . . . . B
A . . . . B
. . . P P B
. . . . C C
. . E . . .
. . E D D D

Step 4:
A . . . . B
A . . . . B
. . . P P B
. . . C C .
. . E . . .
. . E D D D

Step 5:
A . . . . .
A . . . . B
. . . P P B
. . . C C B
. . E . . .
. . E D D D

Step 6:
A . . . . .
A . . . . .
. . . P P B
. . . C C B
. . E . . B

. . E D D D

Step 7:
A . . . . .
A . . . . .
. . . P P B
. . E C C B
. . E . . B
. . . D D D

Step 8:
A . . . . .
A . . . . .
. . . P P B
. . E C C B
. . E . . B
. . D D D .

Step 9:
A . . . . .
A . . . . .
. . . P P .
. . E C C B
. . E . . B
. . D D D B

Step 10:
A . . . . .
A . . . . .
. . . . P P
. . E C C B
. . E . . B
. . D D D B

Solution animation finished (Time taken: 3242.80 ms)

5.1.1.3 Test Case 3 “level3.txt” Heuristic Blocking Piece

Solution found: 27 moves
Total nodes visited: 326
Time taken: 11992.20 ms

Step 1:
. . A . . E
. . A . . E
P P A . . .
. B . . D .
. B C C D .
. . . . D .

Step 2:
. . . . . E
. . A . . E
P P A . . .
. B A . D .
. B C C D .
. . . . D .

Step 3:
. . . . . E
. . A . . E
P P A . D .
. B A . D .
. B C C D .
. . . . . .

Step 4:
. . . . . E
. . A . D E
P P A . D .
. B A . D .
. B C C . .
. . . . . .

Step 5:
. . . . . E
. . A . D E
P P A . D .
. B A . D .
. B . C C .
. . . . . .

Step 6:
. . . . . E
. . . . D E
P P A . D .
. B A . D .
. B A C C .
. . . . . .

Step 7:
. . . . . E
. . . . D E
P P . . D .
. B A . D .
. B A C C .
. . A . . .

Step 8:
. . . . . E
. . . . D E
. P P . D .
. B A . D .
. B A C C .
. . A . . .

Step 9:
. . . . . E
. . . . D E
. . P P D .
. B A . D .
. B A C C .
. . A . . .

Step 10:
. . . . . E
. . . . D E
. B P P D .
. B A . D .
. . A C C .
. . A . . .

Step 11:
. . . . . E
. B . . D E
. B P P D .
. . A . D .
. . A C C .
. . A . . .

Step 12:
. B . . . E
. B . . D E
. . P P D .
. . A . D .
. . A C C .
. . A . . .

Step 13:
. B . . . E
. B . . D E
. P P . D .
. . A . D .
. . A C C .
. . A . . .

Step 14:
. B . . . E
. B . . D E
P P . . D .
. . A . D .
. . A C C .
. . A . . .

Step 15:
. B . . . E
. B . . D E
P P A . D .
. . A . D .
. . A C C .
. . . . . .

Step 16:
. B . . . E
. B A . D E
P P A . D .
. . A . D .
. . . C C .
. . . . . .

Step 17:
. B . . . E
. B A . D E
P P A . D .
. . A . D .
. . C C . .
. . . . . .

Step 18:
. B . . . E
. B A . . E
P P A . D .
. . A . D .
. . C C D .
. . . . . .

Step 19:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
. . C C D .
. . . . D .

Step 20:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
. C C . D .
. . . . D .

Step 21:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
C C . . D .
. . . . D .

Step 22:
. B . . . E
. B . . . E
P P A . . .
. . A . D .
C C A . D .
. . . . D .

Step 23:
. B . . . E
. B . . . E
P P . . . .
. . A . D .
C C A . D .
. . A . D .

Step 24:
. B . . . E
. B . . . E
. P P . . .
. . A . D .
C C A . D .
. . A . D .

Step 25:
. B . . . E
. B . . . E
. . P P . .
. . A . D .
C C A . D .
. . A . D .

Step 26:
. B . . . E
. B . . . E
. . . P P .
. . A . D .
C C A . D .
. . A . D .

Step 27:
. B . . . E
. B . . . E
. . . . P P
. . A . D .
C C A . D .
. . A . D .

Solution animation finished (Time taken: 11992.20 ms)

5.1.1.4 Test Case 4 “level4.txt” Heuristic Blocking Piece

Solution found: 11 moves
Total nodes visited: 34
Time taken: 163.80 ms
Step 1:
. . B B B .
. . A . . E
P P A . . E
. . . . . E
. . C C D D
. . . . . .

Step 2:
. . . B B B
. . A . . E
P P A . . E
. . . . . E
. . C C D D
. . . . . .

Step 3:
. . A B B B
. . A . . E
P P . . . E

. . . . . E
. . C C D D
. . . . . .

Step 4:
. . A B B B
. . A . . E
P P . . . E
. . . . . E
. C C . D D
. . . . . .

Step 5:
. . A B B B
. . A . . E
P P . . . E
. . . . . E
. C C D D .
. . . . . .

Step 6:
. . A B B B
. . A . . .
P P . . . E
. . . . . E
. C C D D E
. . . . . .

Step 7:
. . A B B B
. . A . . .
P P . . . .
. . . . . E
. C C D D E
. . . . . E

Step 8:
. . A B B B
. . A . . .
. P P . . .
. . . . . E
. C C D D E
. . . . . E

Step 9:
. . A B B B
. . A . . .
. . P P . .
. . . . . E
. C C D D E
. . . . . E

Step 10:
. . A B B B
. . A . . .
. . . P P .
. . . . . E
. C C D D E
. . . . . E

Step 11:
. . A B B B
. . A . . .
. . . . P P
. . . . . E
. C C D D E
. . . . . E

Solution animation finished (Time taken: 163.80 ms)

5.1.2 Algoritma Uniform Cost Search

5.1.2.1 Test Case 1 “level1.txt”

Solution found: 12 moves
Total nodes visited: 173
Time taken: 2862.30 ms
Step 1:

. . . . . .
. . . . A .
. . P P A .
. . . B A .
. . . B C C
. . . . . .

Step 2:
. . . . . .
. . . . A .

. P P . A .
. . . B A .
. . . B C C
. . . . . .

Step 3:
. . . . . .
. . . . A .
. P P B A .
. . . B A .
. . . . C C
. . . . . .

Step 4:
. . . . . .
. . . B A .
. P P B A .
. . . . A .
. . . . C C
. . . . . .

Step 5:
. . . B . .
. . . B A .
. P P . A .
. . . . A .
. . . . C C
. . . . . .

Step 6:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . . . C C
. . . . . .

Step 7:
. . . B . .
. . . B A .

. . P P A .
. . . . A .
. . . C C .
. . . . . .

Step 8:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . C C . .
. . . . . .

Step 9:
. . . B . .
. . . B . .
. . P P A .
. . . . A .
. . C C A .
. . . . . .

Step 10:
. . . B . .
. . . B . .
. . P P . .
. . . . A .
. . C C A .
. . . . A .

Step 11:
. . . B . .
. . . B . .
. . . P P .
. . . . A .
. . C C A .
. . . . A .

Step 12:
. . . B . .
. . . B . .

. . . . P P
. . . . A .
. . C C A .
. . . . A .

Solution animation finished (Time taken: 2862.30 ms)

5.1.2.2 Test Case 2 “level2.txt”

Solution found: 10 moves
Total nodes visited: 828
Time taken: 106159.90 ms
Step 1:
A . . . . B
A . . . . B
. P P . . B
. . . . C C
. . E . . .
. . E D D D

Step 2:
A . . . . B
A . . . . B
. . P P . B
. . . . C C
. . E . . .
. . E D D D

Step 3:
A . . . . B
A . . . . B
. . . P P B
. . . . C C
. . E . . .
. . E D D D

Step 4:
A . . . . B
A . . . . B
. . . P P B
. . . C C .

. . E . . .
. . E D D D

Step 5:
A . . . . .
A . . . . B
. . . P P B
. . . C C B
. . E . . .
. . E D D D

Step 6:
A . . . . .
A . . . . .
. . . P P B
. . . C C B
. . E . . B
. . E D D D

Step 7:
A . . . . .
A . . . . .
. . . P P B
. . E C C B
. . E . . B
. . . D D D

Step 8:
A . . . . .
A . . . . .
. . . P P B
. . E C C B
. . E . . B
. . D D D .

Step 9:
A . . . . .
A . . . . .
. . . P P .
. . E C C B

. . E . . B
. . D D D B

Step 10:
A . . . . .
A . . . . .
. . . . P P
. . E C C B
. . E . . B
. . D D D B

Solution animation finished (Time taken: 106159.90 ms)

5.1.2.3 Test Case 3 “level3.txt”

Solution found: 27 moves
Total nodes visited: 633
Time taken: 67228.00 ms
Step 1:
. . . . . .
. . A . . E
P P A . . E
. B A . D .
. B C C D .
. . . . D .

Step 2:
. . . . . E
. . A . . E
P P A . . .
. B A . D .
. B C C D .
. . . . D .

Step 3:
. . . . . E
. . A . . E
P P A . D .
. B A . D .
. B C C D .
. . . . . .

Step 4:

. . . . . E

. . A . D E

P P A . D .

. B A . D .

. B C C . .

. . . . . .

Step 5:

. . . . . E

. . A . D E

P P A . D .

. B A . D .

. B . C C .

. . . . . .

Step 6:

. . . . . E

. . . . D E

P P A . D .

. B A . D .

. B A C C .

. . . . . .

Step 7:

. . . . . E

. . . . D E

P P . . D .

. B A . D .

. B A C C .

. . A . . .

Step 8:

. . . . . E

. . . . D E

. P P . D .

. B A . D .

. B A C C .

. . A . . .

Step 9:
. . . . . E
. . . . D E
. . P P D .
. B A . D .
. B A C C .
. . A . . .

Step 10:
. . . . . E
. . . . D E
. B P P D .
. B A . D .
. . A C C .
. . A . . .

Step 11:
. . . . . E
. B . . D E
. B P P D .
. . A . D .
. . A C C .
. . A . . .

Step 12:
. B . . . E
. B . . D E
. . P P D .
. . A . D .
. . A C C .
. . A . . .

Step 13:
. B . . . E
. B . . D E
. P P . D .
. . A . D .
. . A C C .
. . A . . .

Step 14:
. B . . . E
. B . . D E
P P . . D .
. . A . D .
. . A C C .
. . A . . .

Step 15:
. B . . . E
. B . . D E
P P A . D .
. . A . D .
. . A C C .
. . . . . .

Step 16:
. B . . . E
. B A . D E
P P A . D .
. . A . D .
. . . C C .
. . . . . .

Step 17:
. B . . . E
. B A . D E
P P A . D .
. . A . D .
. . C C . .
. . . . . .

Step 18:
. B . . . E
. B A . . E
P P A . D .
. . A . D .
. . C C D .
. . . . . .

Step 19:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
. . C C D .
. . . . D .

Step 20:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
. C C . D .
. . . . D .

Step 21:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
C C . . D .
. . . . D .

Step 22:
. B . . . E
. B . . . E
P P A . . .
. . A . D .
C C A . D .
. . . . D .

Step 23:
. B . . . E
. B . . . E
P P . . . .
. . A . D .
C C A . D .
. . A . D .

Step 24:
. B . . . E
. B . . . E
. P P . . .
. . A . D .
C C A . D .
. . A . D .

Step 25:
. B . . . E
. B . . . E
. . P P . .
. . A . D .
C C A . D .
. . A . D .

Step 26:
. B . . . E
. B . . . E
. . . P P .
. . A . D .
C C A . D .
. . A . D .

Step 27:
. B . . . E
. B . . . E
. . . . P P
. . A . D .
C C A . D .
. . A . D .

Solution animation finished (Time taken: 67228.00 ms)

5.1.2.4 Test Case 4 “level4.txt”

Solution found: 11 moves
Total nodes visited: 393
Time taken: 27321.80 ms
Step 1:

. . B B B .
. . A . . E
P P A . . E
. . . . . E
. . C C D D
. . . . . .

Step 2:
. . . B B B
. . A . . E
P P A . . E
. . . . . E
. . C C D D
. . . . . .

Step 3:
. . A B B B
. . A . . E
P P . . . E
. . . . . E
. . C C D D
. . . . . .

Step 4:
. . A B B B
. . A . . E
. P P . . E
. . . . . E
. . C C D D
. . . . . .

Step 5:
. . A B B B
. . A . . E
. . P P . E
. . . . . E
. . C C D D
. . . . . .

Step 6:

. . A B B B
. . A . . E
. . . P P E
. . . . . E
. . C C D D
. . . . . .

Step 7:
. . A B B B
. . A . . E
. . . P P E
. . . . . E
. C C . D D
. . . . . .

Step 8:
. . A B B B
. . A . . E
. . . P P E
. . . . . E
. C C D D .
. . . . . .

Step 9:
. . A B B B
. . A . . .
. . . P P E
. . . . . E
. C C D D E
. . . . . .

Step 10:
. . A B B B
. . A . . .
. . . P P .
. . . . . E
. C C D D E
. . . . . E

Step 11:

. . A B B B
. . A . . .
. . . . P P
. . . . . E
. C C D D E
. . . . . E

Solution animation finished (Time taken: 27321.80 ms)

5.1.3 Algoritma A*

5.1.3.1 Test Case 1 “level1.txt” Heuristic Manhattan Distance

Solution found: 12 moves
Total nodes visited: 130
Time taken: 1577.40 ms
Step 1:
. . . . A .
. . . . A .
. P P . A .
. . . B . .
. . . B C C
. . . . . .

Step 2:
. . . . A .
. . . . A .
. P P B A .
. . . B . .
. . . . C C
. . . . . .

Step 3:
. . . . A .
. . . B A .
. P P B A .
. . . . . .
. . . . C C
. . . . . .

Step 4:
. . . B A .
. . . B A .
. P P . A .
. . . . . .

. . . . C C
. . . . . .

Step 5:
. . . B A .
. . . B A .
. . P P A .
. . . . . .
. . . . C C
. . . . . .

Step 6:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . . . C C
. . . . . .

Step 7:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . . C C .
. . . . . .

Step 8:
. . . B . .
. . . B A .
. . P P A .
. . . . A .
. . C C . .
. . . . . .

Step 9:
. . . B . .
. . . B . .
. . P P A .
. . . . A .
. . C C A .
. . . . . .

Step 10:

. . . B . .
. . . B . .
. . P P . .
. . . . A .
. . C C A .
. . . . A .

Step 11:
. . . B . .
. . . B . .
. . . P P .
. . . . A .
. . C C A .
. . . . A .

Step 12:
. . . B . .
. . . B . .
. . . . P P
. . . . A .
. . C C A .
. . . . A .

Solution animation finished (Time taken: 1577.40 ms)

5.1.3.2 Test Case 2 “level2.txt” Heuristic Manhattan Distance

Solution found: 10 moves
Total nodes visited: 445
Time taken: 22640.30 ms
Step 1:
A . . . . B
A . . . . B
. P P . . B
. . . . C C
. . E . . .
. . E D D D

Step 2:
A . . . . B
A . . . . B
. . P P . B
. . . . C C
. . E . . .
. . E D D D

Step 3:
A . . . . B
A . . . . B
. . . P P B
. . . . C C
. . E . . .
. . E D D D

Step 4:
A . . . . B
A . . . . B
. . . P P B
. . . C C .
. . E . . .
. . E D D D

Step 5:
A . . . . .
A . . . . B
. . . P P B
. . . C C B
. . E . . .
. . E D D D

Step 6:
A . . . . .
A . . . . .
. . . P P B
. . . C C B
. . E . . B
. . E D D D

Step 7:
A . . . . .
A . . . . .
. . . P P B
. . E C C B
. . E . . B
. . . D D D

Step 8:
A . . . . .
A . . . . .

. . . P P B
. . E C C B
. . E . . B
. . D D D .

Step 9:
A . . . . .
A . . . . .
. . . P P .
. . E C C B
. . E . . B
. . D D D B

Step 10:
A . . . . .
A . . . . .
. . . . P P
. . E C C B
. . E . . B
. . D D D B

Solution animation finished (Time taken: 22640.30 ms)

5.1.3.3 Test Case 3 “level3.txt” Heuristic Blocking Piece

Solution found: 27 moves
Total nodes visited: 631
Time taken: 34417.70 ms
Step 1:
. . A . . E
. . A . . E
P P A . . .
. B . . D .
. B C C D .
. . . . D .

Step 2:
. . . . . E
. . A . . E
P P A . . .
. B A . D .
. B C C D .
. . . . D .

Step 3:
. . . . . E
. . A . . E
P P A . D .
. B A . D .
. B C C D .
. . . . . .

Step 4:
. . . . . E
. . A . D E
P P A . D .
. B A . D .
. B C C . .
. . . . . .

Step 5:
. . . . . E
. . A . D E
P P A . D .
. B A . D .
. B . C C .
. . . . . .

Step 6:
. . . . . E
. . . . D E
P P A . D .
. B A . D .
. B A C C .
. . . . . .

Step 7:
. . . . . E
. . . . D E
P P . . D .
. B A . D .
. B A C C .
. . A . . .

Step 8:
. . . . . E
. . . . D E
. P P . D .

. B A . D .
. B A C C .
. . A . . .

Step 9:
. . . . . E
. . . . D E
. . P P D .
. B A . D .
. B A C C .
. . A . . .

Step 10:
. . . . . E
. . . . D E
. B P P D .
. B A . D .
. . A C C .
. . A . . .

Step 11:
. . . . . E
. B . . D E
. B P P D .
. . A . D .
. . A C C .
. . A . . .

Step 12:
. B . . . E
. B . . D E
. . P P D .
. . A . D .
. . A C C .
. . A . . .

Step 13:
. B . . . E
. B . . D E
. P P . D .
. . A . D .
. . A C C .
. . A . . .

Step 14:
. B . . . E
. B . . D E
P P . . D .
. . A . D .
. . A C C .
. . A . . .

Step 15:
. B . . . E
. B . . D E
P P A . D .
. . A . D .
. . A C C .
. . . . . .

Step 16:
. B . . . E
. B A . D E
P P A . D .
. . A . D .
. . . C C .
. . . . . .

Step 17:
. B . . . E
. B A . D E
P P A . D .
. . A . D .
. . C C . .
. . . . . .

Step 18:
. B . . . E
. B A . . E
P P A . D .
. . A . D .
. . C C D .
. . . . . .

Step 19:
. B . . . E
. B A . . E
P P A . . .

. . A . D .
. . C C D .
. . . . D .

Step 20:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
. C C . D .
. . . . D .

Step 21:
. B . . . E
. B A . . E
P P A . . .
. . A . D .
C C . . D .
. . . . D .

Step 22:
. B . . . E
. B . . . E
P P A . . .
. . A . D .
C C A . D .
. . . . D .

Step 23:
. B . . . E
. B . . . E
P P . . . .
. . A . D .
C C A . D .
. . A . D .

Step 24:
. B . . . E
. B . . . E
. P P . . .
. . A . D .
C C A . D .
. . A . D .

Step 25:
. B . . . E
. B . . . E
. . P P . .
. . A . D .
C C A . D .
. . A . D .

Step 26:
. B . . . E
. B . . . E
. . . P P .
. . A . D .
C C A . D .
. . A . D .

Step 27:
. B . . . E
. B . . . E
. . . . P P
. . A . D .
C C A . D .
. . A . D .

Solution animation finished (Time taken: 34417.70 ms)

5.1.3.4 Test Case 4 “level4.txt” Heuristic Blocking Piece

Solution found: 11 moves
Total nodes visited: 354
Time taken: 15717.90 ms
Step 1:
. . B B B .
. . A . . E
P P A . . E
. . . . . E
. . C C D D
. . . . . .

Step 2:
. . . B B B
. . A . . E
P P A . . E
. . . . . E
. . C C D D

. . . . . .

Step 3:
. . A B B B
. . A . . E
P P . . . E
. . . . . E
. . C C D D
. . . . . .

Step 4:
. . A B B B
. . A . . E
P P . . . E
. . . . . E
. C C . D D
. . . . . .

Step 5:
. . A B B B
. . A . . E
P P . . . E
. . . . . E
. C C D D .
. . . . . .

Step 6:
. . A B B B
. . A . . .
P P . . . E
. . . . . E
. C C D D E
. . . . . .

Step 7:
. . A B B B
. . A . . .
P P . . . .
. . . . . E
. C C D D E
. . . . . E

Step 8:
. . A B B B

. . A . . .
. P P . . .
. . . . . E
. C C D D E
. . . . . E

Step 9:
. . A B B B
. . A . . .
. . P P . .
. . . . . E
. C C D D E
. . . . . E

Step 10:
. . A B B B
. . A . . .
. . . P P .
. . . . . E
. C C D D E
. . . . . E

Step 11:
. . A B B B
. . A . . .
. . . . P P
. . . . . E
. C C D D E
. . . . . E

Solution animation finished (Time taken: 15717.90 ms)

5.1.4 Algoritma Iterative Deepening Depth-First Search (IDDFS)

5.1.4.1 Test Case 1 “level1.txt”

Melebihi 1000 node, pencarian gagal

5.1.4.2 Test Case 2 “level2.txt”

Melebihi 1000 node, pencarian gagal

5.1.4.3 Test Case 3 “level3.txt”

Melebihi 1000 node, pencarian gagal

5.1.4.4 Test Case 4 “level4.txt”

Melebihi 1000 node, pencarian gagal

5.2 Analisis Pengujian

5.2.1 Analisis Uji Algoritma Greedy Best-First Search

Dari hasil uji, algoritma ini menunjukkan performa terbaik dalam pencarian, dimana level1 diselesaikan dengan 28 node visited, level2 diselesaikan dengan 154 node visited, level3 diselesaikan dengan 326 node visited, dan level4 diselesaikan dengan 34 node visited. Performa unggul ini dapat dikaitkan dengan penggunaan heuristik yang efektif, seperti Blocking Pieces atau Manhattan Distance, yang memungkinkan algoritma untuk memprioritaskan langkah-langkah yang lebih menjanjikan menuju solusi, sehingga mengurangi jumlah node yang perlu dieksplorasi. Namun, efisiensi ini bergantung pada kualitas heuristik yang digunakan, dan dalam kasus tertentu, algoritma ini dapat terjebak pada solusi lokal jika heuristik tidak cukup akurat.

5.2.2 Analisis Uji Algoritma Uniform Cost Search

Dari hasil uji, algoritma ini menunjukkan performa yang kurang efisien dalam pencarian, dimana level1 diselesaikan dengan 173 node visited, level2 diselesaikan dengan 828 node visited, level3 diselesaikan dengan 633 node visited, dan level4 diselesaikan dengan 393 node visited. Ketidakefisienan ini disebabkan oleh pendekatan algoritma yang mengeksplorasi semua node berdasarkan biaya aktual tanpa memanfaatkan heuristik, sehingga menghasilkan ruang pencarian yang lebih luas dan memakan waktu lebih lama untuk mencapai solusi. Perbedaan signifikan dalam jumlah node visited menunjukkan bahwa algoritma ini kurang optimal untuk masalah kompleks seperti Rush Hour, terutama pada level dengan konfigurasi yang lebih menantang.

5.2.3 Analisis Uji Algoritma A*

Dari hasil uji, algoritma ini menunjukkan performa yang cukup efisien dalam pencarian, dimana level1 diselesaikan dengan 130 node visited, level2 diselesaikan dengan 445 node visited, level3 diselesaikan dengan 631 node visited, dan level4 diselesaikan dengan 354 node visited. Efisiensi ini diperoleh dari kombinasi biaya aktual dan heuristik yang admissible, yang memungkinkan algoritma untuk menyeimbangkan eksplorasi dan eksploitasi dalam ruang pencarian.

Meskipun performanya lebih baik daripada Uniform Cost Search, jumlah node visited yang relatif tinggi pada level3 dan level4 menunjukkan bahwa akurasi heuristik yang digunakan dapat menjadi faktor penentu dalam meningkatkan efisiensi lebih lanjut pada konfigurasi yang lebih kompleks.

5.2.4 Analisis Uji Algoritma Iterative Deepening Depth-First Search (IDDFS)

Dari hasil uji, algoritma ini sangat tidak efisien dalam mencari solusi dari inputan masalah game Rush Hour. Karena algoritma ini hanyalah gabungan dari BFS dan DFS, dan tidak memiliki heuristik tambahan sama sekali, IDDFS cenderung mengulang eksplorasi pada kedalaman yang sama berulang kali, yang menghasilkan overhead komputasi yang signifikan. Ketidakefisienan ini diperparah oleh sifat ruang pencarian Rush Hour yang memiliki banyak cabang, sehingga algoritma ini kurang cocok untuk masalah dengan kompleksitas tinggi tanpa dukungan heuristik yang dapat memandu pencarian menuju solusi optimal.

BAB 6 KESIMPULAN DAN SARAN

6.1 Kesimpulan

Proyek Rush Hour Solver pada tugas kecil ini mengimplementasikan beberapa algoritma pathfinding, yaitu Greedy Best-First Search, Uniform Cost Search (UCS), A*, dan Iterative Deepening Depth-First Search (IDDFS), untuk menyelesaikan berbagai level permainan Rush Hour. Hasil uji menunjukkan bahwa Greedy Best-First Search memiliki performa terbaik dengan jumlah node visited yang paling rendah (28, 154, 326, dan 34 untuk level 1 hingga 4), diikuti oleh A* yang cukup efisien (130, 445, 631, dan 354 node visited). Sebaliknya, UCS dan IDDFS menunjukkan performa yang kurang efisien, dengan UCS membutuhkan lebih banyak node visited (173, 828, 633, dan 393) dan IDDFS terbukti sangat tidak efisien karena tidak menggunakan heuristik.

6.2 Saran

Untuk meningkatkan efisiensi program ini, disarankan untuk mengembangkan heuristik yang lebih canggih, seperti kombinasi Blocking Pieces dan Manhattan Distance, guna mengurangi jumlah node visited pada algoritma seperti A* dan Greedy Best-First Search, terutama pada level yang lebih kompleks.

Program ini juga belum mengimplementasikan validasi input, sehingga pastikan input yang diberikan memiliki konfigurasi yang tepat pada inputan .txt.

6.3 Refleksi

Pengerjaan proyek ini memberikan pemahaman yang mendalam tentang penerapan algoritma pencarian dalam menyelesaikan masalah permainan seperti Rush Hour, serta pentingnya heuristik dalam meningkatkan efisiensi pencarian. Saya juga banyak belajar mengenai bahasa pemograman JavaScript dalam proses pengerjaan tugas ini.

LAMPIRAN

Tautan Repository GitHub:

Link Repository : https://github.com/jandhiesto/Tucil3_13523127

Tabel Pengecekan Fitur:

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	✔	

DAFTAR PUSTAKA

Munir, R. (2025). *Penentuan rute (Route/Path Planning) - Bagian 1*. Diakses pada 18 Mei 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf

Munir, R. (2025). *Penentuan rute (Route/Path Planning) - Bagian 2*. Diakses pada 18 Mei 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf

GeeksforGeeks. (n.d.). *Uniform Cost Search (Dijkstra for large Graphs)*. Diakses pada tanggal 18 Mei 2025 dari https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/

GeeksforGeeks. (n.d.). *Greedy Best-First Search Algorithm*. Diakses pada tanggal 18 Mei 2025 dari https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/

Codecademy. (n.d.). *A* Search Algorithm. Diakses pada tanggal 18 Mei 2025 dari https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search

GeeksforGeeks. (n.d.). *Iterative Deepening Search(IDS)/Iterative Deepening Depth First Search(IDDFS)*. Diakses pada tanggal 18 Mei 2025 dari https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/