Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

Disusun oleh:

Ziyan Agil Nur Ramadhan

13622076

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

DAFTAR ISI

Bab 1
Algoritma Penentuan Rute

1. Uniform Cost Search (UCS)
   Algoritma UCS merupakan salah satu bagian dari algoritma Dijkstra yang mencari jalur dengan total biaya terkecil dari simpul awal ke simpul saat ini (g(n)). UCS menjamin solusi optimal jika biaya edge antar simpul tidak negatif. Berikut merupakan langkah-langkah algoritma UCS.
   (a) Inisialisasi priority queue dengan simpul awal, prioritas = g(start) = 0.
   (b) Loop hingga priority queue kosong atau tujuan telah dicapai.
   (c) Keluarkan simpul n dengan nilai g(n) terendah.
   (d) Jika n merupakan suatu tujuan, hentikan dan kembalikan jalur.
   (e) Jika n belum pernah dikunjungi dengan biaya kurang dari atau sama dengan g(n), tandai n sebagai dikunjungi dengan catatan biaya g(n).
   (f) Untuk setiap tetangga m dari n, hitung g(m) = g(n) + biaya dari n ke m dam masukkan m ke priority queue.

2. Greedy Best First Search (GBFS)
   Algoritma GBFS menggunakan heuristik h(n) yang merupakan estimasi biaya dari simpul n menuju simpul tujuan. Mirip seperti algoritma greedy, algoritma GBFS memilih simpul berikutnya tanpa memerhatikan biaya jalur sejauh ini. Berikut merupakan langkah-langkah dalam penyelesaian menggunakan algoritma GBFS.
   (a) Inisialisasi priority queue dengan simpul awal, prioritas = h(start).
   (b) Loop hingga priority queue kosong atau tujuan telah dicapai.
   (c) Keluarkan simpul n dengan nilai h(n) terendah.
   (d) Jika n merupakan suatu tujuan, hentikan.
   (e) Tandai n sebagai dikunjungi.
   (f) Untuk setiap tetangga m jika belum dikunjungi, hitung h(m) dan masukkan ke priority queue.

3. A*
   Algoritma ini menggabungkan antara algoritma UCS dengan algoritma GBFS yaitu dengan menjumlahkan biaya jalur sejauh ini dan estimasi ke tujuan (f(n)=g(n)+h(n)). Langkah-langkah algoritma A* adalah sebagai berikut.
   (a) Inisialisasi priority queue dengan simpul awal, prioritas = f(start) = h(start).
   (b) Loop hingga priority queue kosong.
   (c) Keluarkan simpul n dengan nilai h(n) terendah.
   (d) Jika n merupakan suatu tujuan, hentikan.
   (e) Jika belum dikunjungi atau ditemukan g(n) lebih kecil, tandai dikunjungi dengan biaya g(n).
   (f) Untuk setiap tetangga m jika belum dikunjungi, hitung g(m) = g(n) + biaya dari n ke m dan f(m) = g(m)+h(m), kemudian masukkan ke priority queue.

Bab 2
Implementasi Algoritma

Dalam permainan Rush Hour, tujuan utama adalah memindahkan kendaraan utama (primary piece) ke pintu keluar (exit) dengan menggeser kendaraan-kendaraan lain yang menghalangi jalur menuju pintu keluar. Untuk menyelesaikan permainan ini secara otomatis, diperlukan algoritma pathfinding yang dapat menemukan urutan langkah optimal dari konfigurasi awal hingga konfigurasi akhir. Implementasi ini menggunakan tiga algoritma pathfinding yaitu, Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A* Search (A*).

Sesuai dengan yang telah dijelaskan sebelumnya, algoritma penentuan rute akan menggunakan fungsi-fungsi seperti g(n) dan h(n) yang masing-masing akan dijelaskan lebih rinci di bawah ini.

- g(n) merupakan fungsi yang mengembalikan biaya dari kondisi awal menuju simpul saat ini (n). Dalam implementasi Rush Hour, g(n) direpresentasikan oleh ‘state.cost’ yang merupakan jumlah langkah yang dibutuhkan untuk mencapai kondisi saat ini dari kondisi awal. Fungsi ini biasanya digunakan untuk algoritma UCS (f(n) = g(n)).
- h(n) merupakan fungsi heuristik yang mengestimasi biaya dari simpul saat ini (n) menuju kondisi tujuan. Dalam implementasi ini, h(n) diimplementasikan dengan dua jenis heuristik yaitu ‘manhattanDistance’ yang menghitung jarak Manhattan dari kepingan utama ke pintu keluar serta ‘blockingVehicles’ yang menghitung jumlah kendaraan yang menghalangi jalur kepingan utama menuju pintu keluar. Fungsi ini biasa digunakan untuk algortima GBFS (f(n) = h(n)).
- f(n) merupakan fungsi evaluasi total yang menggabungkan g(n) dan h(n). Nilai f(n) digunakan sebagai dasar prioritas ekspansi simpul dalam algoritma A*.

Dalam implementasi algoritma Heuristik Manhattan Distance pada permainan Rush Hour, heuristik tersebut akan menghitung jarak Manhattan (jumlah langkah horizontal atau vertikal) dari primary piece ke pintu keluar. Heuristik ini bersifat admissible karena jarak Manhattan merupakan batas bawah dari jumlah langkah minimum yang dibutuhkan untuk memindahkan primary piece ke pintu keluar dalam kondisi tanpa adanya piece yang lain selain primary piece.

Selain Heuristik Manhattan Distance, Heuristik Blocking Vehicles diimplementasikan untuk menghitung jumlah kendaraan yang menghalangi jalur primary piece ke pintu keluar. Heuristik ini juga bersifat admissible karena setiap kendaraan yang menghalangi primary piece membutuhkan setidaknya satu langkah untuk dipindahkan, sehingga keadaan final tidak mungkin dicapai dengan langkah kurang dari jumlah kendaraan penghalang.

Dalam permainan Rush Hour, algoritma UCS dan BFS (Breadth-First Search) dapat menghasilkan jalur dan urutan pengembangan simpul yang sama dengan syarat sebagai berikut.

(a) Setiap langkah perpindahan kendaraan memiliki biaya yang sama (dalam implementasi ini, setiap langkah memiliki biaya = 1).

(b) Graf kondisi ruang tidak bersifat siklik (tidak ada kondisi yang dikunjungi lebih dari sekali).

Jika kondisi di atas terpenuhi, algoritma UCS pada dasarnya akan bertindak seperti BFS karena akan memprioritaskan simpul dengan biaya terkecil, yang berarti simpul-simpul akan dieksplorasi berdasarkan kedalaman (level) dalam graf pencarian. Tetapi, dalam implementasi Rush Hour ini, urutan pengembangan simpul tidak selalu sama karena struktur data Priority Queue yang digunakan untuk frontier mungkin memiliki perilaku berbeda dalam menangani simpul-simpul dengan prioritas yang sama serta implementasi UCS menggunakan visited berbasis Map dan menyimpan biaya, sedangkan BFS murni menggunakan Set yang hanya menyimpan kondisi tanpa mempertimbangkan biaya.

Secara teoritis, algoritma A* lebih efisien jika dibandingkan dengan UCS karena A* menggunakan informasi heuristik (h(n)) untuk mengarahkan pencarian ke arah kondisi tujuan, sehingga dapat mengurangi jumlah node yang perlu dieksplorasi. Berbeda dengan UCS yang tidak menggunakan informasi heuristik dan akan menjelajahi kondisi pada ruang berdasarkan biaya terkecil saja, yang bisa menyebabkan pencarian menyebar ke berbagai arah. Tetapi, terdapat kasus di mana A* akan memiliki efisiensi yang sama dengan UCS jika heuristik selalu mengembalikan nilai 0.

Algoritma GBFS hanya akan mempertimbangkan fungsi heuristik h(n) dan mengabaikan biaya yang telah dikeluarkan g(n). Algoritma ini akan selalu memilih simpul yang diperkirakan paling dekat dengan kondisi tujuan berdasarkan heuristik. Hal ini dapat menyebabkan lokal optima di mana algoritma memilih jalur yang terlihat dekat namun pada akhirnya membutuhkan lebih banyak langkah. Algoritma ini bisa menghasilkan solusi yang tidak optimal jika terdapat jalur yang awalnya terlihat jauh dari tujuan tetapi sebenarnya merupakan jalur terpendek. Dalam permainan Rush Hour, algoritma ini hanya mempertimbangkan jarak primary piece ke pintu keluar atau jumlah kendaraan penghalang tidak selalu merefleksikan kompleksitas sebenarnya dalam mencapai kondisi tujuan. Untuk mendapatkan solusi optimal, terkadang perlu menjauhkan suatu kondisi ke kondisi tujuan yang mungkin diabaikan oleh algoritma GBFS. Jadi, algoritma ini tidak menjamin solusi optimal.

Bab 3
Kode Program

Berikut merupakan kode program utama untuk algoritma penyelesaian Rush Hour dengan UCS, GBFS, dan A*.

1. UCS

```
class UcsSolver extends Solver {
    /**
     * @param {GameState} initialState
     * @returns {Object}
     */
    solve(initialState) {
        this.reset();
        const startTime = performance.now();

        const frontier = new PriorityQueue((a, b) => a.cost -
b.cost);
        frontier.push(initialState);

        // Map to track visited states
        const visited = new Map();

        while (!frontier.isEmpty()) {
            const currentState = frontier.pop();
            this.nodesVisited++;

            if (currentState.isGoal()) {
                const endTime = performance.now();
                return this.formatSolution(currentState, endTime -
startTime);
            }

            const stateHash = currentState.getHash();

            if (visited.has(stateHash) && visited.get(stateHash) <=
currentState.cost) {
                continue;
            }

            visited.set(stateHash, currentState.cost);

            const nextStates = currentState.getNextStates();
```

```
            nextStates.forEach(nextState => {
                frontier.push(nextState);
            });
        }

        const endTime = performance.now();
        return this.formatSolution(null, endTime - startTime);
    }
}
```

Implementasi UCS menggunakan priority queue dengan prioritas berdasarkan biaya menjamin solusi optimal karena menjelajahi keadaaan dengan biaya terendah terlebih dahulu. Penggunaan visited.get(stateHash) <= currentState.cost memastikan hanya memproses keadaan yang memiliki biaya lebih rendah dari yang sudah ditemukan. Kompleksitas waktu pada algoritma ini adalah $O(b^d(\log(b^d)))$ dengan b adalah faktor percabangan dan d adalah kedalaman solusi. Dalam kasus terburuk, UCS mungkin harus mengeksplorasi semua kemungkinan keadaan sebelum menemukan solusi, yang bisa mencapai $O(b^d)$ simpul. Setiap operasi pada priority queue memerlukan waktu O(log N). Dalam kasus terburuk, N bisa mencapai b^d.

2. GBFS

```
class GreedySolver extends Solver {
    /**
     * @param {Function}
     */
    constructor(heuristicFunction) {
        super();
        this.heuristicFunction = heuristicFunction;
    }

    /**
     * @param {GameState} initialState
     * @returns {Object}
     */
    solve(initialState) {
        this.reset();
        const startTime = performance.now();
        const frontier = new PriorityQueue((a, b) => {
            const heuristicA = this.heuristicFunction(a);
            const heuristicB = this.heuristicFunction(b);
            return heuristicA - heuristicB;
        });

        frontier.push(initialState);
```

```
        const visited = new Set();

        while (!frontier.isEmpty()) {
            const currentState = frontier.pop();
            this.nodesVisited++;

            if (currentState.isGoal()) {
                const endTime = performance.now();
                return this.formatSolution(currentState, endTime -
startTime);
            }

            const stateHash = currentState.getHash();

            if (visited.has(stateHash)) {
                continue;
            }

            visited.add(stateHash);

            const nextStates = currentState.getNextStates();

            nextStates.forEach(nextState => {
                frontier.push(nextState);
            });
        }

        const endTime = performance.now();
        return this.formatSolution(null, endTime - startTime);
    }
}
```

Algoritma GBFS memiliki prioritas murni berdasarkan nilai heuristik, mengabaikan biaya perjalanan sejauh ini. Algoritma ini menggunakan Set untuk keadaan yang telah dikunjungi, lebih sederhana dari UCS karena tidak perlu membandingkan biaya. Simpul yang dijelajahi umumnya lebih sedikit dari UCS tetapi tidak menjamin solusi optimal. Kompleksitas waktu algoritma UCS adalah $O(b^m(\log(b^m)))$ dengan b adalah faktor percabangan dan m adalah kedalaman maksimum dari ruang pencarian. GBFS berpotensi mengeksplorasi jalur yang sangat dalam jika heuristik mengarahkannya ke arah yang salah, hingga mencapai kedalaman maksimum m. Setiap operasi pada priority queue memerlukan waktu O(log N) dengan N bisa mencapai kompleksitas $O((\log(b^m))$.

3. A*

```
class AStarSolver extends Solver {
    /**
```

```
     * @param {Function} heuristicFunction
     */
    constructor(heuristicFunction) {
        super();
        this.heuristicFunction = heuristicFunction;
    }

    /**
     * @param {GameState} initialState
     * @returns {Object}
     */
    solve(initialState) {
        this.reset();
        const startTime = performance.now();

        const frontier = new PriorityQueue((a, b) => {
            const heuristicA = a.cost + this.heuristicFunction(a);
            const heuristicB = b.cost + this.heuristicFunction(b);
            return heuristicA - heuristicB;
        });

        frontier.push(initialState);

        const visited = new Map();

        while (!frontier.isEmpty()) {
            const currentState = frontier.pop();
            this.nodesVisited++;

            if (currentState.isGoal()) {
                const endTime = performance.now();
                return this.formatSolution(currentState, endTime -
startTime);
            }
            const stateHash = currentState.getHash();

            if (visited.has(stateHash) && visited.get(stateHash) <=
currentState.cost) {
                continue;
            }

            visited.set(stateHash, currentState.cost);
```

```
            const nextStates = currentState.getNextStates();

            nextStates.forEach(nextState => {
                frontier.push(nextState);
            });
        }

        const endTime = performance.now();
        return this.formatSolution(null, endTime - startTime);
    }
}
```

Algoritma A* menggabungkan kelebihan UCS (memperhitungkan biaya sejauh ini) dan GBFS (menggunakan heuristik). Fungsi evaluasinya adalah f(n) = g(n) + h(n), di mana g(n) adalah biaya sejauh ini dan h(n) adalah estimasi biaya untuk sampai tujuan. Algoritma ini menjamin solusi optimal jika heuristik yang digunakan admissible. Kompleksitas waktu pada algoritma ini adalah $O(b^d(\log(b^d)))$ dengan b adalah faktor percabangan, d adalah kedalaman solusi dan $\log(b^d)$ adalah kompleksitas operasi priority queue.

4. Manhattan Distance

```
static manhattanDistance(state) {
        const board = state.board;
        const primaryPiece = board.primaryPiece;
        const exitPosition = board.exitPosition;

        if (!primaryPiece || !exitPosition) {
            return Infinity;
        }
        let primaryX, primaryY;

        if (primaryPiece.orientation === 'horizontal') {
            if (exitPosition.x === board.width - 1) {
                primaryX = primaryPiece.x + primaryPiece.length - 1;
            } else {
                primaryX = primaryPiece.x;
            }
            primaryY = primaryPiece.y;
        } else {
            primaryX = primaryPiece.x;
            if (exitPosition.y === board.height - 1) {
                primaryY = primaryPiece.y + primaryPiece.length - 1;
            } else {
                primaryY = primaryPiece.y;
            }
```

```
        }
        return Math.abs(primaryX - exitPosition.x) +
Math.abs(primaryY - exitPosition.y);
    }
```

Algoritma heuristik ini menghitung jarak Manhattan (jarak dalam koordinat x + y) dari kendaraan utama ke pintu keluar. Kompleksitas waktu dari algoritma heuristik ini yaitu O(1) karena hanya melakukan perhitungan sederhana yang tidak tergantung pada ukuran board atau jumlah kendaraan.

5. Blocking Piece

```
static blockingVehicles(state) {
        const board = state.board;
        const primaryPiece = board.primaryPiece;
        const exitPosition = board.exitPosition;

        if (!primaryPiece || !exitPosition) {
            return Infinity;
        }
        let blockingCount = 0;

        if (primaryPiece.orientation === 'horizontal') {
            const y = primaryPiece.y;

            if (exitPosition.x > primaryPiece.x) {
                for (let x = primaryPiece.x + primaryPiece.length; x
< exitPosition.x; x++) {
                    if (board.grid[y][x].isOccupied()) {
                        blockingCount++;
                    }
                }
            } else {
                for (let x = primaryPiece.x - 1; x > exitPosition.x;
x--) {
                    if (board.grid[y][x].isOccupied()) {
                        blockingCount++;
                    }
                }
            }
        } else {
            const x = primaryPiece.x;

            if (exitPosition.y > primaryPiece.y) {
```

```
                for (let y = primaryPiece.y + primaryPiece.length; y
< exitPosition.y; y++) {
                    if (board.grid[y][x].isOccupied()) {
                        blockingCount++;
                    }
                }
            } else {
                for (let y = primaryPiece.y - 1; y > exitPosition.y;
y--) {
                    if (board.grid[y][x].isOccupied()) {
                        blockingCount++;
                    }
                }
            }
        }
        return blockingCount;
    }
```

Algoritma heuristik ini menghitung berapa banyak kendaraan yang menghalangi jalan langsung dari kendaraan utama ke pintu keluar. Kompleksitas waktu algoritma ini adalah O(n) di mana n adalah jarak dari kendaraan utama ke pintu keluar. Fungsi akan melakukan iterasi sepanjang jalur dari kendaraan utama menuju pintu keluar yang berpotensi sampai sebesar ukuran board (dalam kasus terburuk).

No.	Modul	Atribut/Metode	Keterangan
1.	Board.js	width dan height	Dimensi papan
		grid	Array 2D berisi Cell
		pieces	Map berisi objek Piece
		primaryPiece	Referensi ke kendaraan utama
		exitPosition	Koordinat pintu keluar
		placePiece()	Menempatkan kendaraan pada papan
		setExit()	Menentukan posisi pintu keluar
		isValidMove()	Mengecek apakah suatu gerakan valid
		movePiece()	Memindahkan kendaraan
		isSolved()	Mengecek apakah permainan telah selesai
		getPossibleMoves()	Mendapatkan semua gerakan yang mungkin
		clone()	Membuat salinan board

2.	Cell.js	occupiedBy	ID kendaraan yang menempati sel (null jika kosong)
		isOccupied()	Mengecek apakah sel ditempati
		occupy()	Menandai sel sebagai ditempati
		clear()	Mengosongkan sel
3.	Piece.js	id	Identifikasi unik kendaraan
		x dan y	Koordinat posisi awal
		length	Panjang kendaraan
		orientation	Orientasi kendaraan (horizontal/vertical)
		isPrimary	Penanda kendaraan utama
		getCells()	Mendapatkan semua sel yang ditempati
		getNewCellsAfterMove()	Mendapatkan sel-sel setelah pergerakan
		move()	Memindahkan kendaraan
		clone()	Membuat salinan kendaraan
4.	GameState.js	board	Objek Board
		moveHistory	Riwayat gerakan
		cost	Jumlah gerakan yang telah dilakukan
		getHash()	Mendapatkan representasi string unik dari state
		getNextStates()	Mendapatkan semua state yang mungkin dicapai
		isGoal()	Mengecek apakah state merupakan goal state
		clone()	Membuat salinan state
5.	Solver.js	nodeVisited	Jumlah node yang dikunjungi
		solve()	Metode abstrak yang harus diimplementasikan subclass
		reset()	Mengatur ulang counter
		formatSolution()	Memformat hasil pencarian
6.	UcsSolver.js	solve()	Implementasi algoritma Uniform Cost Search

7.	GreedySolver.js	solve()	Implementasi algoritma Greedy Best First Search
8.	AStarSolver.js	solve()	Implementasi algoritma A* Search
9.	Heuristics.js	manhattanDistance()	Menghitung jarak Manhattan dari primary piece ke exit
		blockingVehicles()	Menghitung jumlah kendaraan penghalang
		getHeuristics()	Factory method untuk mendapatkan fungsi heuristik
10.	FileReader.js	readPuzzleFile()	Membaca file konfigurasi permainan
11.	PriorityQueue.js	this.comparator	Fungsi pembanding untuk menentukan urutan heap (min-heap)
		this.heap	Array internal yang menyimpan elemen-elemen heap
		size()	Mengembalikan jumlah elemen di dalam heap
		isEmpty()	Mengembalikan apakah heap kosong
		peek()	Mengembalikan elemen dengan prioritas tertinggi tanpa menghapusnya
		push(value)	Menambahkan value ke heap dan melakukan penyesuaian
		pop	Menghapus dan mengembalikan elemen prioritas tertinggi
		_siftUp()	Menjaga invariant heap setelah push, memindahkan node baru ke posisi yang benar
		_siftDown()	Menjaga invariant heap setelah pop, memindahkan elemen pengganti root ke posisi yang benar
12.	Visualizer.js	COLORS	Objek yang menyimpan fungsi pewarnaan (chalk) untuk: PRIMARY_PIECE, EXIT, MOVED_PIECE, DEFAULT
		displayBoard(board, movedPieceId)	Mencetak representasi ASCII board di console, dengan highlight untuk potongan tertentu (movedPieceId)
		displaySolution(initialState, movesHistory)	Menunjukkan langkah demi langkah: papan awal, setiap gerakan, hingga akhir

		formatDirection(dir)	Mengonversi kode gerakan
		displayStats({ executionTime, movesHistory })	Mencetak statistik run: waktu eksekusi dan jumlah total langkah
13.	index.js	main()	Berisi fungsi main untuk menjalankan program

Bab 4
Implementasi dan Pengujian

No.	Input	Output
1.	Enter path to puzzle file: test/1.txt Initial Board: CCCC... PPA....K ..A.... .BBB... Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 1	Solving with Uniform Cost Search... --- Solution --- Papan Awal CCCC... PPA....K ..A.... .BBB... Gerakan 1: B-kanan CCCC... PPA....K ..A.... ..BBB.. Gerakan 2: B-kanan CCCC... PPA....K ..A.... ...BBB. Gerakan 3: A-bawah CCCC... PP.....K ..A.... ..ABBB. Gerakan 4: P-kanan CCCC... .PP....K ..A.... ..ABBB. Gerakan 5: P-kanan CCCC... ..PP...K ..A.... ..ABBB. Gerakan 6: P-kanan CCCC... ...PP..K ..A.... ..ABBB. Gerakan 7: P-kanan CCCC... ....PP.K ..A.... ..ABBB. Gerakan 8: P-kanan CCCC... .....PPK ..A.... ..ABBB. Game solved! --------- Statistcs --------- Execution time: 5.33 ms Total moves: 8 ------------------------------

2.	Enter path to puzzle file: test/1.txt Initial Board: CCCC... PPA....K ..A.... .BBB... Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 2 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Solving with Greedy Best First Search using manhattan heuristic... --- Solution --- Papan Awal CCCC... PPA....K ..A.... .BBB... Gerakan 1: B-kanan CCCC... PPA....K ..A.... ..BBB.. Gerakan 2: B-kanan CCCC... PPA....K ..A.... ...BBB. Gerakan 3: B-kanan CCCC... PPA....K ..A.... ....BBB Gerakan 4: C-kanan .CCCC.. PPA....K ..A.... ....BBB Gerakan 5: C-kanan ..CCCC. PPA....K ..A.... ....BBB Gerakan 6: C-kanan ...CCCC PPA....K ..A.... ....BBB Gerakan 7: B-kiri ...CCCC PPA....K ..A.... ...BBB. Gerakan 8: A-bawah ...CCCC PP.....K ..A.... ..ABBB. Gerakan 9: P-kanan ...CCCC .PP....K ..A.... ..ABBB. Gerakan 10: P-kanan ...CCCC ..PP...K ..A.... ..ABBB. Gerakan 11: P-kanan ...CCCC ...PP..K ..A....

		..ABBB. Gerakan 12: P-kanan ...CCCC ....PP.K ..A.... ..ABBB. Gerakan 13: P-kanan ...CCCC .....PPK ..A.... ..ABBB. Game solved! --------- Statistcs --------- Execution time: 3.56 ms Total moves: 13 ----------------------------
3.	Enter path to puzzle file: test/1.txt Initial Board: CCCC... PPA....K ..A.... .BBB... Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 3 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Solving with A* Search using manhattan heuristic... --- Solution --- Papan Awal CCCC... PPA....K ..A.... .BBB... Gerakan 1: B-kanan CCCC... PPA....K ..A.... ..BBB.. Gerakan 2: B-kanan CCCC... PPA....K ..A.... ...BBB. Gerakan 3: A-bawah CCCC... PP.....K ..A.... ..ABBB. Gerakan 4: P-kanan CCCC... .PP....K ..A.... ..ABBB. Gerakan 5: P-kanan CCCC... ..PP...K ..A.... ..ABBB. Gerakan 6: P-kanan CCCC... ...PP..K ..A.... ..ABBB. Gerakan 7: P-kanan CCCC... ....PP.K ..A.... ..ABBB. Gerakan 8: P-kanan

		CCCC... .....PPK ..A.... ..ABBB. Game solved! --------- Statistcs --------- Execution time: 2.72 ms Total moves: 8 ----------
4.	Enter path to puzzle file: test/2.txt Initial Board: CCCC... K..A.PPP ..A.... .BBB... Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 1	Solving with Uniform Cost Search... --- Solution --- Papan Awal CCCC... K..A.PPP ..A.... .BBB... Gerakan 1: B-kanan CCCC... K..A.PPP ..A.... ..BBB.. Gerakan 2: B-kanan CCCC... K..A.PPP ..A.... ...BBB. Gerakan 3: A-bawah CCCC... K....PPP ..A.... ..ABBB. Gerakan 4: P-kiri CCCC... K...PPP. ..A.... ..ABBB. Gerakan 5: P-kiri CCCC... K..PPP.. ..A.... ..ABBB. Gerakan 6: P-kiri CCCC... K.PPP... ..A.... ..ABBB. Gerakan 7: P-kiri CCCC... KPPP.... ..A.... ..ABBB. Game solved! --------- Statistcs --------- Execution time: 9.51 ms Total moves: 7 ----------
5.	Enter path to puzzle file: test/2.txt	Solving with Greedy Best First Search using manhattan heuristic... --- Solution ---

	Initial Board: CCCC... K..A.PPP ..A.... .BBB... Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 2 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Papan Awal CCCC... K..A.PPP ..A.... .BBB... Gerakan 1: P-kiri CCCC... K..APPP. ..A.... .BBB... Gerakan 2: C-kanan .CCCC.. K..APPP. ..A.... .BBB... Gerakan 3: C-kanan ..CCCC. K..APPP. ..A.... .BBB... Gerakan 4: B-kanan ..CCCC. K..APPP. ..A.... ..BBB.. Gerakan 5: B-kanan ..CCCC. K..APPP. ..A.... ...BBB. Gerakan 6: B-kanan ..CCCC. K..APPP. ..A.... ....BBB Gerakan 7: A-bawah ..CCCC. K...PPP. ..A.... ..A.BBB Gerakan 8: P-kiri ..CCCC. K..PPP.. ..A.... ..A.BBB Gerakan 9: P-kiri ..CCCC. K.PPP... ..A.... ..A.BBB Gerakan 10: P-kiri ..CCCC. KPPP.... ..A.... ..A.BBB Game solved! --------- Statistcs --------- Execution time: 3.14 ms Total moves: 10 ------------------------------

6.	Enter path to puzzle file: test/2.txt Initial Board: CCCC... K..A.PPP ..A.... .BBB... Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 3 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Solving with A* Search using manhattan heuristic... --- Solution --- Papan Awal CCCC... K..A.PPP ..A.... .BBB... Gerakan 1: B-kanan CCCC... K..A.PPP ..A.... ..BBB.. Gerakan 2: B-kanan CCCC... K..A.PPP ..A.... ...BBB. Gerakan 3: P-kiri CCCC... K..APPP. ..A.... ...BBB. Gerakan 4: A-bawah CCCC... K...PPP. ..A.... ..ABBB. Gerakan 5: P-kiri CCCC... K..PPP.. ..A.... ..ABBB. Gerakan 6: P-kiri CCCC... K.PPP... ..A.... ..ABBB. Gerakan 7: P-kiri CCCC... KPPP.... ..A.... ..ABBB. Game solved! --------- Statistcs --------- Execution time: 3.94 ms Total moves: 7 ----------
7.	Enter path to puzzle file: test/3.txt Initial Board: AA.PGG BBBP.H CDD.IH C...IH C.FJJJ EEF.LL K	Solving with Uniform Cost Search... --- Solution --- Papan Awal AA.PGG BBBP.H CDD.IH C...IH C.FJJJ EEF.LL K Gerakan 1: F-atas AA.PGG BBBP.H

	Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 1	CDD.IH C.F.IH C.FJJJ EE..LL K Gerakan 2: P-bawah AA..GG BBBP.H CDDPIH C.F.IH C.FJJJ EE..LL K Gerakan 3: E-kanan AA..GG BBBP.H CDDPIH C.F.IH C.FJJJ .EE.LL K Gerakan 4: C-bawah AA..GG BBBP.H .DDPIH C.F.IH C.FJJJ CEE.LL K Gerakan 5: D-kiri AA..GG BBBP.H DD.PIH C.F.IH C.FJJJ CEE.LL K Gerakan 6: F-atas AA..GG BBBP.H DDFPIH C.F.IH C..JJJ CEE.LL K Gerakan 7: P-bawah AA..GG BBB..H DDFPIH C.FPIH C..JJJ CEE.LL K Gerakan 8: J-kiri AA..GG BBB..H DDFPIH C.FPIH C.JJJ. CEE.LL K Gerakan 9: H-bawah AA..GG BBB... DDFPIH C.FPIH C.JJJH

		CEE.LL K Gerakan 10: B-kanan AA..GG .BBB.. DDFPIH C.FPIH C.JJJH CEE.LL K Gerakan 11: B-kanan AA..GG ..BBB. DDFPIH C.FPIH C.JJJH CEE.LL K Gerakan 12: B-kanan AA..GG ...BBB DDFPIH C.FPIH C.JJJH CEE.LL K Gerakan 13: F-atas AA..GG ..FBBB DDFPIH C..PIH C.JJJH CEE.LL K Gerakan 14: F-atas AAF.GG ..FBBB DD.PIH C..PIH C.JJJH CEE.LL K Gerakan 15: D-kanan AAF.GG ..FBBB .DDPIH C..PIH C.JJJH CEE.LL K Gerakan 16: C-atas AAF.GG ..FBBB CDDPIH C..PIH C.JJJH .EE.LL K Gerakan 17: C-atas AAF.GG C.FBBB CDDPIH C..PIH ..JJJH .EE.LL K

		Gerakan 18: J-kiri AAF.GG C.FBBB CDDPIH C..PIH .JJJ.H .EE.LL K Gerakan 19: J-kiri AAF.GG C.FBBB CDDPIH C..PIH JJJ..H .EE.LL K Gerakan 20: P-bawah AAF.GG C.FBBB CDD.IH C..PIH JJJP.H .EE.LL K Gerakan 21: P-bawah AAF.GG C.FBBB CDD.IH C...IH JJJP.H .EEPLL K Game solved! --------- Statistcs --------- Execution time: 253.45 ms Total moves: 21 ----------
8.	Enter path to puzzle file: test/3.txt Initial Board: AA.PGG BBBP.H CDD.IH C...IH C.FJJJ EEF.LL K Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 2 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	. . . Gerakan 234: J-kanan AAFGG. ..FBBB CDDPIH C..PIH C.JJJH ..EELL K Gerakan 235: G-kanan AAF.GG ..FBBB CDDPIH C..PIH C.JJJH ..EELL K Gerakan 236: J-kiri AAF.GG ..FBBB CDDPIH C..PIH CJJJ.H ..EELL K Gerakan 237: E-kiri

		AAF.GG ..FBBB CDDPIH C..PIH CJJJ.H .EE.LL K Gerakan 238: E-kiri AAF.GG ..FBBB CDDPIH C..PIH CJJJ.H EE..LL K Gerakan 239: L-kiri AAF.GG ..FBBB CDDPIH C..PIH CJJJ.H EE.LL. K Gerakan 240: L-kiri AAF.GG ..FBBB CDDPIH C..PIH CJJJ.H EELL.. K Gerakan 241: G-kiri AAFGG. ..FBBB CDDPIH C..PIH CJJJ.H EELL.. K Gerakan 242: J-kanan AAFGG. ..FBBB CDDPIH C..PIH C.JJJH EELL.. K Gerakan 243: L-kanan AAFGG. ..FBBB CDDPIH C..PIH C.JJJH EE.LL. K Gerakan 244: L-kanan AAFGG. ..FBBB CDDPIH C..PIH C.JJJH EE..LL K Gerakan 245: C-atas AAFGG. C.FBBB CDDPIH

```
C..PIH
..JJJH
EE..LL
   K

Gerakan 246: E-kanan
AAFGG.
C.FBBB
CDDPIH
C..PIH
..JJJH
.EE.LL
   K

Gerakan 247: E-kanan
AAFGG.
C.FBBB
CDDPIH
C..PIH
..JJJH
..EELL
   K

Gerakan 248: G-kanan
AAF.GG
C.FBBB
CDDPIH
C..PIH
..JJJH
..EELL
   K

Gerakan 249: J-kiri
AAF.GG
C.FBBB
CDDPIH
C..PIH
.JJJ.H
..EELL
   K

Gerakan 250: J-kiri
AAF.GG
C.FBBB
CDDPIH
C..PIH
JJJ..H
..EELL
   K

Gerakan 251: P-bawah
AAF.GG
C.FBBB
CDD.IH
C..PIH
JJJP.H
..EELL
   K

Gerakan 252: E-kiri
AAF.GG
C.FBBB
CDD.IH
C..PIH
JJJP.H
.EE.LL
   K

Gerakan 253: P-bawah
AAF.GG
C.FBBB
CDD.IH
C...IH
JJJP.H
.EEPLL
```

		K Game solved! --------- Statistcs --------- Execution time: 279.83 ms Total moves: 253 -----------------------------
9.	Enter path to puzzle file: test/3.txt Initial Board: AA.PGG BBBP.H CDD.IH C...IH C.FJJJ EEF.LL K Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 3 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Solving with A* Search using manhattan heuristic... --- Solution --- Papan Awal AA.PGG BBBP.H CDD.IH C...IH C.FJJJ EEF.LL K Gerakan 1: P-bawah AA..GG BBBP.H CDDPIH C...IH C.FJJJ EEF.LL K Gerakan 2: F-atas AA..GG BBBP.H CDDPIH C.F.IH C.FJJJ EE..LL K Gerakan 3: E-kanan AA..GG BBBP.H CDDPIH C.F.IH C.FJJJ .EE.LL K Gerakan 4: C-bawah AA..GG BBBP.H .DDPIH C.F.IH C.FJJJ CEE.LL K Gerakan 5: D-kiri AA..GG BBBP.H DD.PIH C.F.IH C.FJJJ CEE.LL K Gerakan 6: P-bawah AA..GG BBB..H DD.PIH C.FPIH C.FJJJ CEE.LL K

```
Gerakan 7: F-atas
AA..GG
BBB..H
DDFPIH
C.FPIH
C..JJJ
CEE.LL
   K

Gerakan 8: J-kiri
AA..GG
BBB..H
DDFPIH
C.FPIH
C.JJJ.
CEE.LL
   K

Gerakan 9: H-bawah
AA..GG
BBB...
DDFPIH
C.FPIH
C.JJJH
CEE.LL
   K

Gerakan 10: B-kanan
AA..GG
.BBB..
DDFPIH
C.FPIH
C.JJJH
CEE.LL
   K

Gerakan 11: B-kanan
AA..GG
..BBB.
DDFPIH
C.FPIH
C.JJJH
CEE.LL
   K

Gerakan 12: B-kanan
AA..GG
...BBB
DDFPIH
C.FPIH
C.JJJH
CEE.LL
   K

Gerakan 13: F-atas
AA..GG
..FBBB
DDFPIH
C..PIH
C.JJJH
CEE.LL
   K

Gerakan 14: F-atas
AAF.GG
..FBBB
DD.PIH
C..PIH
C.JJJH
CEE.LL
   K

Gerakan 15: D-kanan
AAF.GG
```

		..FBBB .DDPIH C..PIH C.JJJH CEE.LL K Gerakan 16: C-atas AAF.GG ..FBBB CDDPIH C..PIH C.JJJH .EE.LL K Gerakan 17: C-atas AAF.GG C.FBBB CDDPIH C..PIH ..JJJH .EE.LL K Gerakan 18: J-kiri AAF.GG C.FBBB CDDPIH C..PIH .JJJ.H .EE.LL K Gerakan 19: J-kiri AAF.GG C.FBBB CDDPIH C..PIH JJJ..H .EE.LL K Gerakan 20: P-bawah AAF.GG C.FBBB CDD.IH C..PIH JJJP.H .EE.LL K Gerakan 21: P-bawah AAF.GG C.FBBB CDD.IH C...IH JJJP.H .EEPLL K Game solved! --------- Statistcs --------- Execution time: 258.91 ms Total moves: 21 ------------------------------
10.	Enter path to puzzle file: test/4.txt Initial Board:	Solving with Uniform Cost Search... --- Solution --- Papan Awal AAB..F ..BCDF

	AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 1	GPPCDFK GH.III GHJ... LLJMM. Gerakan 1: D-atas AAB.DF ..BCDF GPPC.FK GH.III GHJ... LLJMM. Gerakan 2: C-atas AABCDF ..BCDF GPP..FK GH.III GHJ... LLJMM. Gerakan 3: I-kiri AABCDF ..BCDF GPP..FK GHIII. GHJ... LLJMM. Gerakan 4: F-bawah AABCD. ..BCDF GPP..FK GHIIIF GHJ... LLJMM. Gerakan 5: F-bawah AABCD. ..BCD. GPP..FK GHIIIF GHJ..F LLJMM. Gerakan 6: F-bawah AABCD. ..BCD. GPP...K GHIIIF GHJ..F LLJMMF Gerakan 7: P-kanan AABCD. ..BCD. 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 Game solved! --------- Statistcs --------- Execution time: 70.23 ms Total moves: 9 -----------------------------
11.	Enter path to puzzle file: test/4.txt Initial Board: AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 2 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Solving with Greedy Best First Search using manhattan heuristic... --- Solution --- Papan Awal AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Gerakan 1: C-atas AABC.F ..BCDF GPP.DFK GH.III GHJ... LLJMM. Gerakan 2: P-kanan AABC.F ..BCDF G.PPDFK GH.III GHJ... LLJMM. Gerakan 3: D-atas AABCDF ..BCDF G.PP.FK GH.III GHJ... LLJMM. Gerakan 4: P-kanan AABCDF ..BCDF G..PPFK GH.III GHJ... LLJMM. Gerakan 5: G-atas AABCDF G.BCDF G..PPFK GH.III .HJ... LLJMM. Gerakan 6: I-kiri AABCDF G.BCDF G..PPFK GHIII. .HJ... LLJMM. Gerakan 7: G-bawah AABCDF ..BCDF G..PPFK GHIII.

		GHJ... LLJMM. Gerakan 8: H-atas AABCDF ..BCDF GH.PPFK GHIII. G.J... LLJMM. Gerakan 9: B-bawah AA.CDF ..BCDF GHBPPFK GHIII. G.J... LLJMM. Gerakan 10: G-atas AA.CDF G.BCDF GHBPPFK GHIII. ..J... LLJMM. Gerakan 11: F-bawah AA.CD. G.BCDF GHBPPFK GHIIIF ..J... LLJMM. Gerakan 12: M-kanan AA.CD. G.BCDF GHBPPFK GHIIIF ..J... LLJ.MM Gerakan 13: A-kanan .AACD. G.BCDF GHBPPFK GHIIIF ..J... LLJ.MM Gerakan 14: H-bawah .AACD. G.BCDF G.BPPFK GHIIIF .HJ... LLJ.MM Gerakan 15: A-kiri AA.CD. G.BCDF G.BPPFK GHIIIF .HJ... LLJ.MM Gerakan 16: G-bawah AA.CD. ..BCDF G.BPPFK GHIIIF GHJ... LLJ.MM

		Gerakan 17: M-kiri AA.CD. ..BCDF G.BPPFK GHIIIF GHJ... LLJMM. Gerakan 18: F-bawah AA.CD. ..BCD. G.BPPFK GHIIIF GHJ..F LLJMM. Gerakan 19: M-kanan AA.CD. ..BCD. G.BPPFK GHIIIF GHJ..F LLJ.MM Gerakan 20: A-kanan .AACD. ..BCD. G.BPPFK GHIIIF GHJ..F LLJ.MM Gerakan 21: M-kiri .AACD. ..BCD. G.BPPFK GHIIIF GHJ..F LLJMM. Gerakan 22: F-bawah .AACD. ..BCD. G.BPP.K GHIIIF GHJ..F LLJMMF Gerakan 23: P-kanan .AACD. ..BCD. G.B.PPK GHIIIF GHJ..F LLJMMF Game solved! --------- Statistcs --------- Execution time: 22.39 ms Total moves: 23 ----------
12.	Enter path to puzzle file: test/4.txt Initial Board: AAB..F ..BCDF GPPCDFK GH.III	Solving with A* Search using manhattan heuristic... --- Solution --- Papan Awal AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.

	GHJ... LLJMM. Choose algorithm (1: UCS, 2: Greedy Best First Search, 3: A*): 3 Choose heuristic (1: Manhattan Distance, 2: Blocking Vehicles): 1	Gerakan 1: I-kiri AAB..F ..BCDF GPPCDFK GHIII. GHJ... LLJMM. Gerakan 2: F-bawah AAB... ..BCDF GPPCDFK GHIIIF GHJ... LLJMM. Gerakan 3: F-bawah AAB... ..BCD. GPPCDFK GHIIIF GHJ..F LLJMM. Gerakan 4: F-bawah AAB... ..BCD. GPPCD.K GHIIIF GHJ..F LLJMMF Gerakan 5: C-atas AABC.. ..BCD. GPP.D.K GHIIIF GHJ..F LLJMMF Gerakan 6: D-atas AABCD. ..BCD. GPP...K GHIIIF GHJ..F LLJMMF Gerakan 7: P-kanan AABCD. ..BCD. 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 Game solved! --------- Statistcs ---------

		Execution time: 43.17 ms Total moves: 9 ----------

LAMPIRAN

Tautan *repository* GitHub : https://github.com/ziyanagil/Tucil3_13622076

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

DAFTAR PUSTAKA

Munir, R. (2025). *Penentuan rute (Route/Path Planning) - Bagian 1*. Homepage Rinaldi Munir. Retrieved 20 May, 2025, from 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*. Homepage Rinaldi Munir. Retrieved 20 May, 2025, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf