Laporan Tugas Kecil 3

Syahrizal Bani Khairan - 13523063

Untuk kuliah IF2211 Strategi Algoritma

21 Mei 2025

1. Identifikasi Masalah

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.

Oleh karena itu, komponen dalam permasalahan ini adalah:

1. **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
2. **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 –** adalah tempat *primary piece* dapat digerakkan keluar untuk menyelesaikan permainan. Hanya *primary piece* yang dapat keluar dari papan.
5. **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

Dengan ini, solusi permainan dapat dicari dengan menggunakan algoritma pathfinding. Setiap konfigurasi papan dengan penempatan piece berbeda dapat dianggap sebagai suatu simpul dalam graf. Jika piece dapat digerakkan secara valid hingga *state* papan berubah, dikatakan kedua simpul *state* sebelum dan sesudah pergerakan adalah 2 simpul yang bertetangga. Dengan diketahuinya kondisi awal dan kondisi target berupa kondisi dimana

primary piece dapat keluar dari papan, dapat digunakan algoritma traversal graf khususnya algoritma pathfinding seperti A*.

2. Spesifikasi

Algoritma pathfinding dalam graf adalah metode untuk menemukan jalur dari satu simpul ke simpul lain dalam suatu graf. Pencarian jalur dapat membutuhkan optimasi berupa rute terpendek, misal dalam graf berbobot jumlah bobot semua sisi dalam jalur atau dalam graf tak berbobot jumlah simpul pada jalur. Pada permasalahan ini, setiap sisi dalam graf memiliki bobot sama (tidak ada perbedaan untuk semua gerakan yang valid) dan secara efektif berupa graf tak berbobot.

Algoritma pathfinding dapat dibagi menjadi 2 kategori, yaitu *informed* dan *uninformed*. Algoritma *uninformed* tidak memiliki pengetahuan tambahan mengenai simpul *goal*, melainkan hanya bergantung pada pengetahuan terhadap simpul-simpul yang sudah ditelusuri pada saat itu juga. Sebaliknya, algoritma *informed* memiliki pengetahuan tambahan, misalnya dengan bantuan heuristik, untuk membantu melakukan pencarian yang mengarah secara langsung ke simpul *goal*.

Pohon status dinamis yang dibangkitkan selama eksekusi algoritma berupa pohon yang pada setiap simpulnya memiliki keseluruhan status terkait simpul tersebut, dan juga memuat informasi simpul *parent* yang membangkitkannya. Pada awalnya, pohon dinamis hanya memiliki satu simpul untuk kondisi awal. Lalu setiap simpul daun akan diproses untuk membangkitkan simpul anak dalam pohon dinamis yang berkorespondensi dengan simpul tetangga dalam graf. Jika dibutuhkan optimasi, perlu adanya pemilihan simpul yang akan diproses agar tidak membangkitkan terlalu banyak simpul dan algoritma dapat memberikan hasil berupa jalur optimal. Algoritma yang diimplementasi untuk tugas ini adalah Uniform Cost Search (UCS), greedy best first search, dan A*.

2.1. Algoritma Uniform Cost Search (UCS)

UCS (Uniform Cost Search) adalah algoritma pencarian jalur (pathfinding) yang digunakan untuk menemukan jalur dengan biaya(cost) total terkecil dari simpul awal ke simpul tujuan dalam sebuah graf berbobot. UCS merupakan varian dari algoritma Dijkstra, dan dapat dianggap sebagai algoritma Best-First Search yang menggunakan total biaya dari simpul awal sebagai fungsi heuristik.

Pembangkitan simpul dalam pohon status dinamis akan didahulukan pada simpul yang memiliki biaya terhadap simpul asal terkecil. Algoritma ini termasuk kategori *uninformed*, karena hanya menggunakan informasi yang ditemukan selama penelusuran.

2.2. Algoritma greedy best first search

Greedy Best-First Search (GBFS) adalah algoritma pencarian jalur (pathfinding) yang memilih simpul untuk dikunjungi berikutnya berdasarkan seberapa dekat simpul tersebut ke

tujuan, menurut suatu fungsi heuristik. Tujuannya adalah mencapai simpul akhir secepat mungkin, tanpa memperhitungkan biaya total jalur. Adanya heuristik untuk mencari simpul *goal* menyebabkan algoritma ini terkategori sebagai algoritma *informed.* Fungsi heuristik yang baik seharusnya mengevaluai suatu simpul sehingga bernilai lebih kecil untuk simpul yang lebih dekat dengan simpul *goal*. Harapannya, dengan memproses simpul dengan evaluasi fungsi heuristik rendah dapat ditemukan simpul *goal*.

2.3. Algoritma A*

Algoritma A* adalah algoritma pencarian jalur yang menggabungkan kelebihan dari Uniform Cost Search (UCS) dan Greedy Best-First Search (GBFS). Tujuannya adalah menemukan jalur optimal dari simpul awal ke simpul tujuan, dengan konsiderasi biaya dari simpul awal ke simpul saat ini dan heuristik untuk mengestimasi biaya dari simpul saat ini ke simpul *goal*. A* bekerja dengan memilih simpul untuk dikunjungi berikutnya berdasarkan fungsi evaluasi:

$$f(n) = g(n) + h(n)$$

dengan $f(n)$ adalah total biaya suatu simpul, $g(n)$ adalah biaya dari simpul tertentu dari simpul awal, dan $h(n)$ adalah estimasi biaya dari simpul tertentu ke simpul *goal* berdasarkan suatu heuristik. Seperti algoritma sebelumnya, simpul daun yang terlebih dahulu diproses adalah simpul yang memiliki nilai f terkecil dari semua simpul daun, karena f adalah estimasi jalur dengan biaya optimal dari simpul awal ke simpul *goal*.

Algoritma A* menghasilkan jalur optimal jika menggunakan heuristik *admissible*. Heuristik yang *admissible* adalah heuristik yang tidak melakukan overestimasi dibanding biaya optimal nyata. [1]

Algoritma UCS dan greedy best first search juga dapat dikatakan sebagai kasus khusus algoritma A*. Algoritma UCS adalah algoritma A* yang tidak memiliki fungsi heuristik ($h(n) = 0$ untuk semua simpul) sedangkan greedy best first search tidak menggunakan biaya dari simpul awal ($g(n) = 0$ untuk semua simpul).

3. Implementasi

Implementasi menggunakan bahasa JavaScript. Kode program dapat dijalankan misalnya dengan runtime environment NodeJS.

3.1. Main[.]js

```
import { createInterface } from 'readline';
import { writeFileSync } from 'fs';
import { ReadConfig } from './ConfigParser.js';
import { Pathfind } from './Pathfind.js';
import { distanceToDoor, nodesFromStart, piecesInFront, recursiveBlockers } from
```

```
'./Heuristics.js';

const SEARCH_ALGORITHMS = {
    "1": "A*",
    "2": "UCS",
    "3": "Greedy",
}
const HEURISTICS = {
    "1": {func: piecesInFront, desc: "Jumlah piece antara primary piece dan pintu"},
    "2": {func: distanceToDoor, desc: "Jarak primary piece ke pintu"},
}

function askQuestion(rl, query) {
    return new Promise(resolve => rl.question(query, (answer) => resolve(answer)));
}

async function main() {
    const rl = createInterface({
        input: process.stdin,
        output: process.stdout
    });

    const filename = await askQuestion(rl, "Nama file konfigurasi: ");
    let algoMsg = "Pilih algoritma:\n|";
    for (const [key, value] of Object.entries(SEARCH_ALGORITHMS)) {
        algoMsg += ` ${key}: ${value}`.padEnd(10, " ") + "|";
    }
    const algo = await askQuestion(rl, algoMsg + "\n");
    let heuristicMsg = "Pilih heuristic:\n";
    for (const [key, value] of Object.entries(HEURISTICS)) {
        heuristicMsg += ` ${key}: ${value.desc}` + "\n";
    }
    if (!SEARCH_ALGORITHMS[algo]) {
        console.error("Invalid algorithm choice");
        rl.close();
        return;
    }
    if (SEARCH_ALGORITHMS[algo] !== "UCS") {
        var heuristic = await askQuestion(rl, heuristicMsg);
        if (!HEURISTICS[heuristic]) {
            console.error("Invalid heuristic choice");
            rl.close();
            return;
        }
    }
```

```
    let puzzleState = null;
    try {
        puzzleState = ReadConfig(filename);
    } catch (error) {
        console.error("Error reading config file\n", error);
        rl.close();
        return;
    }


    let start, end;
    let goalNode = null;
    switch (algo) {
        case "1":
            console.log("Running A* algorithm...");
            puzzleState.nodeCount = 0;

            start = performance.now();
            goalNode = Pathfind(puzzleState, nodesFromStart,
HEURISTICS[heuristic].func);
            end = performance.now();
            break;
        case "2":
            console.log("Running UCS algorithm...");
            puzzleState.nodeCount = 0;

            start = performance.now();
            goalNode = Pathfind(puzzleState, nodesFromStart, null);
            end = performance.now();
            break;
        case "3":
            console.log("Running Greedy algorithm...");
            puzzleState.nodeCount = 0;

            start = performance.now();
            goalNode = Pathfind(puzzleState, null, HEURISTICS[heuristic].func);
            end = performance.now();
            break;
        default:
            console.error("Invalid algorithm choice");
            rl.close();
            return;
    }

    console.log("Total nodes visited: " + puzzleState.nodeCount);
    console.log("Time taken: " + (end - start) + " ms");
```

```
    if (goalNode !== null) {
        puzzleState.printPath(goalNode);
    } else {
        console.log("No solution found.");
    }

    const savePath = await askQuestion(rl, "Simpan ke file? (y/n): ");
    if (savePath.toLowerCase() === 'y') {
        const fileName = await askQuestion(rl, "Nama file : ");
        try {
            let pathStrings = [];
            pathStrings.push("Total nodes visited: " + puzzleState.nodeCount +
"\n");
            pathStrings.push("Time taken: " + (end - start) + " ms\n");
            pathStrings.push("Algoritma: " + SEARCH_ALGORITHMS[algo] + "\n");
            if (heuristic !== undefined) {
                pathStrings.push("Heuristic: " + HEURISTICS[heuristic].desc + "\n");
            }
            pathStrings.push(puzzleState.printPath(goalNode, true));
            writeFileSync(fileName, pathStrings.join(""));

            console.log("Solusi disimpan ke " + fileName);
        } catch (error) {
            console.error("Error saving path to file\n", error);
        }
    }

    rl.close();
    return;
}

main();
```

3.2. ConfigParser[.]js

Membaca file konfigurasi puzzle. Pengecekan error dalam konfigurasi masih belum dilakukan secara cukup banyak.

```
import { readFileSync } from 'fs';
import { EOL } from 'os';
import { PuzzleState, SIDES, EMPTY_SPACE } from './PuzzleState.js';

export function ReadConfig(filename) { // ret: BoardState
    console.log("Reading config file: " + filename);

    // Load file
    const filePath = filename;
    const fileContent = readFileSync(filePath, 'utf-8');
    const lines = fileContent.split(EOL);

    // Parse file
    let doorSide = SIDES.NONE;
    let doorPos = {x: -1, y: -1};   // door position resides inside the board along
the edges
    let boardWidth = 6;
    let boardHeight = 6;
    let pieceCount = 0;

    let lineIdx = -1;
    let board = [];
    lines.forEach((line) => {
        lineIdx++;

        if (lineIdx === 0) {
            [boardWidth, boardHeight] = line.split(' ').map((x) => {
                try {
                    return parseInt(x);
                } catch (e) {
                    throw new Error("Invalid board size: " + x);
                }
            });
            if (boardWidth < 1 || boardHeight < 1) {
                throw new Error("Invalid board size: " + boardWidth + "x" +
boardHeight);
            }
        } else if (lineIdx === 1) {
            [pieceCount] = line.split(' ').map((x) => {
                try {
                    return parseInt(x);
                } catch (e) {
                    throw new Error("Invalid piece count: " + x);
                }
            });
        } else {
```

```
            if (board.length >= boardHeight && doorSide !== SIDES.NONE) {
                return; // Ignore extra lines
            }

            // Deduce door side here
            if (line.trim() === "K" ) {
                if (doorSide !== SIDES.NONE) {
                    throw new Error("Multiple doors found");
                }

                if (lineIdx === 2) {
                    doorSide = SIDES.TOP;
                    doorPos.x = line.indexOf("K");
                    doorPos.y = 0;
                    lineIdx--;
                } else {
                    doorSide = SIDES.BOTTOM;
                    doorPos.x = line.indexOf("K");
                    doorPos.y = boardHeight - 1;
                }
                line = line.replace("K", " ");
                return;
            }
            if (line.at(0) === "K") {
                if (doorSide !== SIDES.NONE) {
                    throw new Error("Multiple doors found");
                }
                doorSide = SIDES.LEFT;
                doorPos.x = 0;
                doorPos.y = lineIdx - 2;
                line = line.replace("K", " ");
            }
            if (line.at(-1) === "K") {
                if (doorSide !== SIDES.NONE) {
                    throw new Error("Multiple doors found");
                }
                doorSide = SIDES.RIGHT;
                doorPos.x = boardWidth - 1;
                doorPos.y = lineIdx - 2;
                line = line.replace("K", " ");
            }

            line = line.trim();
            if (line.length !== boardWidth) {
                line = line.padEnd(boardWidth, EMPTY_SPACE);
                line = line.slice(0, boardWidth);
```

```
            }
            board.push(line)
        }

        return;
    })

    let state = new PuzzleState(boardWidth, boardHeight, pieceCount, board.join(""),
doorSide, doorPos);
    console.log("Config file read successfully");
    switch (doorSide) {
        case SIDES.TOP:
            console.log("Door side: UP");
            break;
        case SIDES.RIGHT:
            console.log("Door side: RIGHT");
            break;
        case SIDES.BOTTOM:
            console.log("Door side: DOWN");
            break;
        case SIDES.LEFT:
            console.log("Door side: LEFT");
            break;
        default:
            console.log("Door side: NONE");
    }
    console.log("Door position: " + doorPos.x + ", " + doorPos.y);
    console.log("Piece count: " + pieceCount);
    console.log("Board size: " + boardWidth + "x" + boardHeight);
    return state;
}
```

3.3. PuzzleState[.]js

Terdapat beberapa definisi class yang berperan dalam pembangkitan pohon status dinamis. Setiap simpul berisi papan, referensi ke simpul parent, dan gerakan yang dilakukan agar dapat meraih simpul ini dari simpul parent. Terdapat juga implementasi dari priority queue sederhana untuk mengurutkan simpul daun dalam pohon dinamis berdasarkan nilai f.

```
export const EMPTY_SPACE = ".";
export const PRIMARY_PIECE = "P";
export const DOOR = "K";
export const SIDES = {
    TOP: 0,
    RIGHT: 1,
```

```
    BOTTOM: 2,
    LEFT: 3
};

const ANSI_RESET = "\x1b[0m";
const ANSI_BLUE = "\x1b[34m";
const ANSI_GREEN = "\x1b[32m";
const ANSI_RED = "\x1b[31m";

export { SearchNode, PriorityQueue, PuzzleState };

class SearchNode {
    constructor(board, parentNode, letter, moveDistance) {
        this.board = board;
        this.parent = parentNode;
        this.movedLetter = letter;
        this.moveDistance = moveDistance;

        // These values must be set correctly
        this.g = 0;
        this.h = 0;
        this.f = this.g + this.h;
    }

    getF () {
        return this.f;
    }
    setG(g) {
        this.g = g;
        this.f = this.g + this.h;
    }
    setH(h) {
        this.h = h;
        this.f = this.g + this.h;
    }

    getSignature() {
        return this.board.boardStrings().join("");
    }

    toString(count, doorSide, doorPos, colorize = true) {
        const piece = this.board.pieces.get(this.movedLetter);

        let lines = [];
        if (count == 0) {
            lines.push("Papan awal\n");
```

```
        } else {
            lines.push(`Gerakan ${count}: ${this.movedLetter}-`)
            if (this.moveDistance > 0 && piece.isHorizontal) {
                lines.push("kanan\n");
            } else if (this.moveDistance < 0 && piece.isHorizontal) {
                lines.push("kiri\n");
            } else if (this.moveDistance > 0 && !piece.isHorizontal) {
                lines.push("bawah\n");
            } else if (this.moveDistance < 0 && !piece.isHorizontal) {
                lines.push("atas\n");
            }
        }
        // Board
        let boardStrings = this.board.boardStrings();
        if (doorSide === SIDES.TOP) {
            const doorLine = " ".repeat(doorPos.x) + DOOR;
            boardStrings.unshift(doorLine);
        } else if (doorSide === SIDES.BOTTOM) {
            const doorLine = " ".repeat(doorPos.x) + DOOR;
            boardStrings.push(doorLine);
        } else if (doorSide === SIDES.LEFT) {
            for (let i = 0; i < boardStrings.length; i++) {
                if (i === doorPos.y) {
                    boardStrings[i] = DOOR + boardStrings[i];
                } else {
                    boardStrings[i] = " " + boardStrings[i];
                }
            }
        } else if (doorSide === SIDES.RIGHT) {
            boardStrings[doorPos.y] += DOOR;
        }
        // Colorize the board
        if (colorize) {
            for (let i = 0; i < boardStrings.length; i++) {
                let rowchars = boardStrings[i].split("");
                for (let j = 0; j < rowchars.length; j++) {
                    const char = rowchars[j];
                    if (char === PRIMARY_PIECE) {
                        rowchars[j] = ANSI_RED + char + ANSI_RESET;
                    } else if (char === this.movedLetter) {
                        rowchars[j] = ANSI_BLUE + char + ANSI_RESET;
                    } else if (char === DOOR) {
                        rowchars[j] = ANSI_GREEN + char + ANSI_RESET;
                    }
                }
                boardStrings[i] = rowchars.join("");
```

```
            }
        }

        lines.push(boardStrings.join("\n"));
        lines.push("\n");

        return lines.join("");
    }
}

class PriorityQueue {
    // Simple array priority queue implementation
    constructor() {
        this.queue = [];    // Of SearchNode
    }

    enqueue(node) {
        this.queue.push(node);
        this.queue.sort((a, b) => a.getF() - b.getF());
    }

    dequeue() {
        return this.queue.shift();
    }

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

/*
-------------------------------------------------------------------------------
-- */

class PuzzleState {
    constructor(width, height, pieceCount, board, doorSide, doorPos) {
        this.pieceCount = pieceCount;
        this.doorSide = doorSide;
        this.doorPos = doorPos;
        this.initialBoard = new Board(width, height, board);
        this.nodeCount = 0;

    }

    getAllPieces() {
        return this.initialBoard.pieces.keys();
```

```
    }

    generateNode(letter, moveDistance, node) { // Factory to generate a new search
node
        if (!node.board.isMoveValid(letter, moveDistance)) {
            return null;
        }

        let newBoard = new Board(node.board.width, node.board.height,
node.board.board);
        newBoard.movePiece(letter, moveDistance);
        return new SearchNode(newBoard, node, letter, moveDistance);
    }

    isGoalNode(node) {
        const primaryPiece = node.board.pieces.get(PRIMARY_PIECE);
        // Check if the primary piece incide with the door
        for (let i = 0; i < primaryPiece.length; i++) {
            const x = primaryPiece.anchorX + (primaryPiece.isHorizontal ? i : 0);
            const y = primaryPiece.anchorY + (primaryPiece.isHorizontal ? 0 : i);
            if (x === this.doorPos.x && y === this.doorPos.y) {
                return true;
            }
        }
        return false;
    }

    printPath(node, toFile = false) {
        const path = [];
        while (node !== null) {
            path.push(node);
            node = node.parent;
        }
        path.reverse();

        let result = [];
        for (let i = 0; i < path.length; i++) {
            result.push(path[i].toString(i, this.doorSide, this.doorPos, toFile ===
false) + "\n");
        }

        if (toFile) {
            return result.join("");
        } else {
            console.log(result.join(""));
            return;
```

```
        }
    }
}

class Board {
    constructor(width, height, board) {
        this.width = width;
        this.height = height;
        this.board = [];
        for (let i = 0; i < width * height; i++) {
            this.board.push(board.at(i));
        }

        this.pieces = new Map();
        for (let i = 0; i < width * height; i++) {
            const letter = this.pieceAt(i);
            if (!this.pieces.has(letter)) {
                const isHorizontal = true;     // Temporary value
                const length = 1;              // Temporary value
                const xPos = i % width;
                const yPos = Math.floor(i / width);
                this.pieces.set(letter, new Piece(xPos, yPos, length,
isHorizontal));
            } else {
                const piece = this.pieces.get(letter);
                piece.length++;
                const xPos = i % width;
                if (piece.anchorX == xPos) {
                    piece.isHorizontal = false;
                } else {
                    piece.isHorizontal = true;
                }
            }
        }
    }

    pieceAt(x, y = null) {
        if (y === null) {
            return this.board[x];
        }
        return this.board[y*this.width+x];
    }
    movePiece(letter, moveDistance) {
        const piece = this.pieces.get(letter);
        if (piece === undefined) {
            return false;
```

```
        }

        // Clear current piece from board
        for (let i = 0; i < piece.length; i++) {
            const x = piece.anchorX + (piece.isHorizontal ? i : 0);
            const y = piece.anchorY + (piece.isHorizontal ? 0 : i);
            this.board[y * this.width + x] = EMPTY_SPACE;
        }

        if (piece.isHorizontal) {
            piece.anchorX += moveDistance;
        } else {
            piece.anchorY += moveDistance;
        }

        for (let i = 0; i < piece.length; i++) {
            const x = piece.anchorX + (piece.isHorizontal ? i : 0);
            const y = piece.anchorY + (piece.isHorizontal ? 0 : i);
            this.board[y * this.width + x] = letter;
        }

        return true;
    }

    isMoveValid(letter, moveDistance) {
        const piece = this.pieces.get(letter);
        if (piece === undefined) {
            return false;
        }

        for (let i = 0; i < piece.length; i++) {
            const coordX = piece.anchorX + (piece.isHorizontal ? i : 0) +
(piece.isHorizontal ? moveDistance : 0);
            const coordY = piece.anchorY + (piece.isHorizontal ? 0 : i) +
(piece.isHorizontal ? 0 : moveDistance);

            if (coordX < 0 || coordX >= this.width || coordY < 0 || coordY >=
this.height) {
                return false;
            }

            const letterAtCoord = this.pieceAt(coordX, coordY);
            if (letterAtCoord !== EMPTY_SPACE && letterAtCoord !== letter) {
                return false;
            }
        }
```

```
        return true;
    }

    boardStrings() {
        let rows = [];
        for (let i = 0; i < this.height; i++) {
            let row = "";
            for (let j = 0; j < this.width; j++) {
                row += this.pieceAt(j, i);
            }
            rows.push(row);
        }
        return rows;
    }
}

class Piece {
    constructor(anchorX, anchorY, length, isHorizontal) {
        this.anchorX = anchorX;
        this.anchorY = anchorY;
        this.length = length;
        this.isHorizontal = isHorizontal;
    }
}
```

3.4. Pathfind[.]js

Algoritma utama pencarian jalur.

```
import { SearchNode, PriorityQueue } from './PuzzleState.js';

const MAX_ITER = 10000;

// Returns the final goal node if found, otherwise null
// gEstimator and hEstimator are functions that estimates the g(n) and h(n) costs
// If only gEstimator is provided, the pathfinding algorithm is equivalent to UCS
// If only hEstimator is provided, the pathfinding algorithm is equivalent to Greedy
// If both are provided, the pathfinding algorithm is equivalent to A* search
export function Pathfind(puzzleState, gEstimator, hEstimator) {
    const q = new PriorityQueue();
    const visited = new Map();

    const startNode = new SearchNode(puzzleState.initialBoard, null, null, 0);
    if (gEstimator !== null) {
        startNode.setG(gEstimator(startNode, puzzleState));
```

```
    }
    if (hEstimator !== null) {
        startNode.setH(hEstimator(startNode, puzzleState));
    }
    q.enqueue(startNode);
    visited.set(startNode.getSignature(), startNode.getF());

    let finalNode = null;  // Will point to the goal node if found
    let iterCount = 0;
    while (!q.isEmpty()) {
        const node = q.dequeue();
        if (puzzleState.isGoalNode(node)) {
            finalNode = node;
            break;
        }
        if (visited.has(node.getSignature()) && node.getF() >
visited.get(node.getSignature())) {
            continue;
        }

        // Generate nodes by trying to move pieces in this current state
        for (const letter of puzzleState.getAllPieces()) {
            tryMoves(puzzleState, node, q, visited, letter, gEstimator, hEstimator);
        }

        iterCount++;
        if (iterCount > MAX_ITER) {
            console.log("Max iterations reached. Stopping search.");
            break;
        }
    }

    return finalNode;
}

// Try moving pieces forwards and backwards a varying distance
function tryMoves(puzzleState, node, queue, visited, piece, gEstimator,hEstimator) {
    for (const direction of [1, -1]) { // Try positive and negative directions
        let moveDist = 0;
        while (true) {
            puzzleState.nodeCount++;
            moveDist += direction;
            const newNode = puzzleState.generateNode(piece, moveDist, node);
            if (newNode === null) break;

            if (gEstimator !== null) {
```

```
                newNode.setG(gEstimator(newNode, puzzleState));
            }
            if (hEstimator !== null) {
                newNode.setH(hEstimator(newNode, puzzleState));
            }

            const signature = newNode.getSignature();
            if (!visited.has(signature) || newNode.getF() < visited.get(signature))
{
                queue.enqueue(newNode);
                visited.set(signature, newNode.getF());
            }
        }
    }
    return null;
}
```

3.5. Heuristics[.]js

Berisi fungsi estimasi g(n) dan h(n). Cuplikan ini tidak memuat heuristik yang sayangnya tidak dapat diselesaikan saat ini.

```
import { EMPTY_SPACE, PRIMARY_PIECE, SIDES } from "./PuzzleState.js";

/* g(n) estimators */

export function nodesFromStart(node, puzzleState) {
    if (node.parent === null || node.parent === undefined) {
        return 0;
    }
    return node.parent.g + 1;
}

/* h(n) estimators */

// Calculates the distance from the primary piece to the door. Not admissible
export function distanceToDoor(node, puzzleState) {
    switch (puzzleState.doorSide) {
        case SIDES.TOP:
            return node.board.pieces.get(PRIMARY_PIECE).anchorY;
        case SIDES.BOTTOM:
            return node.board.height - (node.board.pieces.get(PRIMARY_PIECE).anchorY
+ node.board.pieces.get(PRIMARY_PIECE).length);
        case SIDES.LEFT:
            return node.board.pieces.get(PRIMARY_PIECE).anchorX;
```

```
        case SIDES.RIGHT:
            return node.board.width - (node.board.pieces.get(PRIMARY_PIECE).anchorX
+ node.board.pieces.get(PRIMARY_PIECE).length);
    }
    return 0; // Invalid case
}

// Count how many pieces are in between the primary piece and the door
export function piecesInFront(node, puzzleState) {
    const coordsToCheck = [];
    const primaryPiece = node.board.pieces.get(PRIMARY_PIECE);
    switch (puzzleState.doorSide) {
        case SIDES.TOP:
            for (let i = primaryPiece.anchorY-1; i >= 0; i--) {
                coordsToCheck.push({ x: primaryPiece.anchorX, y: i });
            }
            break;
        case SIDES.BOTTOM:
            for (let i = primaryPiece.anchorY+primaryPiece.length; i <
node.board.height; i++) {
                coordsToCheck.push({ x: primaryPiece.anchorX, y: i });
            }
            break;
        case SIDES.LEFT:
            for (let i = primaryPiece.anchorX-1; i >= 0; i--) {
                coordsToCheck.push({ x: i, y: primaryPiece.anchorY });
            }
            break;
        case SIDES.RIGHT:
            for (let i = primaryPiece.anchorX+primaryPiece.length; i <
node.board.width; i++) {
                coordsToCheck.push({ x: i, y: primaryPiece.anchorY });
            }
            break;
    }

    let count = 0;
    for (const coord of coordsToCheck) {
        const piece = node.board.pieceAt(coord.x, coord.y);
        if (piece !== null && piece !== PRIMARY_PIECE && piece !== EMPTY_SPACE) {
            count++;
        }
    }
    return count;
}
```

4. Analisis

Algoritma pencarian jalur sangat bergantung pada evaluasi fungsi f(n), g(n), dan h(n) yang mempengaruhi efisiensi pencarian dan sifat optimal pencarian. Pada kasus permasalahan ini, graf secara efektif tak berbobot atau memiliki bobot 1. Fungsi g(n) menghitung banyak simpul yang dilalui untuk menggapai simpul tertentu. Ini menyebabkan algoritma UCS sama dengan algoritma BFS, karena semua simpul dengan kedalaman yang sama memiliki nilai g(n) yang sama dan simpul akan diproses per level kedalaman.

Fungsi heuristik yang digunakan juga dapat mempengaruhi efisiensi pencarian. Fungsi heuristik yang baik dapat memberikan gradien atau 'lembah' untuk simpul-simpul yang mengarah ke simpul *goal* sehingga harapannya pencarian graf tidak memproses simpul tak optimal. Algoritma greedy best first search dengan heuristik yang baik menemukan jalur lebih cepat daripada UCS, tetapi secara teoritis tidak optimal. Karena GBFS mengabaikan biaya dari simpul awal, algoritma tersebut dapat menghasilkan jalur melewati jalur yang mahal ke suatu simpul tetapi diestimasi dekat dengan simpul *goal*.

Algoritma A* menggabungkan keunggulan UCS dan GBFS sebagai algoritma yang optimal (tidak seperti GBFS) dan lebih efisien (dibanding UCS). Fungsi heuristik yang baik yang digunakan pada algoritma A* dapat menjadikannya lebih efisien, tetapi sifat optimal algoritma bergantung pada heuristik yang digunakan. A* menghasilkan jalur optimal jika dan hanya jika fungsi heuristik tidak melakukan overestimasi dibanding biaya optimal yang nyata.

Pada implementasi, heuristik *piecesInFront* adalah heuristik *admissible* dan akan menghasilkan solusi optimal. Agar *primary piece* dapat keluar dari papan, semua piece yang ada di antara *primary piece* dan pintu harus disingkirkan. Untuk menyingkirkannya, perlu dilakukan setidaknya satu kali gerakan untuk setiap *piece*. Jika ada n penghalang, tidak mungkin hanya dilakukan n-1 atau kurang jumlah gerakan sehingga *primary piece* dapat keluar karena masih tersisa setidaknya n-(n-1) penghalang.

Selain itu, heuristik distanceToDoor adalah heuristik yang tidak admissible. Masih dalam situasi sebelumnya, beberapa *piece* yang menghalangi *primary piece* mungkin sudah tersingkirkan. Tetapi, heuristic ini akan memilih untuk menggerakkan *primary piece* menuju pintu meskipun dapat dilakukan penyingkiran *piece* penghalang karena dipercaya lebih dekat menuju simpul *goal*. Ini menyebabkan jumlah gerakan berlebih yang tidak optimal.

5. Hasil Uji

Test Case	Hasil

```
6 6
11
AAB..F
..BCDF
GPPCDFK
GH.III
GH....
LL.MM.
```

Output: res1.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test1.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
1
Pilih heuristic:
 1: Jumlah piece antara primary piece dan pintu
 2: Jarak primary piece ke pintu
1
Reading config file: test/test1.txt
Config file read successfully
Door side: RIGHT
Door position: 5, 2
Piece count: 11
Board size: 6x6
Running A* algorithm...
Total nodes visited: 3904
Time taken: 19.363399999998364 ms
Papan awal
AAB..F
..BCDF
GPPCDFK
GH.III
GH....
LL.MM.
```

Output: res1alt.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test1.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
1
Pilih heuristic:
 1: Jumlah piece antara primary piece dan pintu
 2: Jarak primary piece ke pintu
2
Reading config file: test/test1.txt
Config file read successfully
Door side: RIGHT
Door position: 5, 2
Piece count: 11
Board size: 6x6
Running A* algorithm...
Total nodes visited: 6429
Time taken: 22.64550000000054 ms
Papan awal
AAB..F
..BCDF
GPPCDFK
GH.III
GH....
LL.MM.
```

```
6 6
12
 K
AABCDF
..BCDF
GZZI..
GP.I..
GP.I..
LL.MM.
```

Output: res2.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test2.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
1
Pilih heuristic:
 1: Jumlah piece antara primary piece dan pintu
 2: Jarak primary piece ke pintu
1
Reading config file: test/test2.txt
Config file read successfully
Door side: UP
Door position: 1, 0
Piece count: 12
Board size: 6x6
Running A* algorithm...
Total nodes visited: 41627
Time taken: 91.8068000000003 ms
Papan awal
 K
AABCDF
..BCDF
GZZI..
GP.I..
GP.I..
LL.MM.
```

```
6 6
12
..CDF.
BBCDFG
.PAA.G
HP.I.G
HZZI..
HMM...
 K
```

Output: res3.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test3.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
1
Pilih heuristic:
 1: Jumlah piece antara primary piece dan pintu
 2: Jarak primary piece ke pintu
1
Reading config file: test/test3.txt
Config file read successfully
Door side: DOWN
Door position: 1, 5
Piece count: 12
Board size: 6x6
Running A* algorithm...
Total nodes visited: 21172
Time taken: 44.460799999999836 ms
Papan awal
..CDF.
BBCDFG
.PAA.G
HP.I.G
HZZI..
HMM...
 K

Gerakan 1: M-kanan
```

```
6 6
12
 ..BCDF
 ..BCDF
 G..ZZ.
KGHPPII
 GH...Z
 ..MM.Z
```

Output: res4.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test4.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
1
Pilih heuristic:
 1: Jumlah piece antara primary piece dan pintu
 2: Jarak primary piece ke pintu
1
Reading config file: test/test4.txt
Config file read successfully
Door side: LEFT
Door position: 0, 3
Piece count: 12
Board size: 6x6
Running A* algorithm...
Total nodes visited: 2417
Time taken: 12.623700000000099 ms
Papan awal
 ..BCDF
 ..BCDF
 G..ZZ.
KGHPPII
 GH...Z
 ..MM.Z

Gerakan 1: G-atas
 G.BCDF
 G.BCDF
 G..ZZ.
K.HPPII
 .H...Z
 ..MM.Z
```

```
6 6
3
....B.
....B.
..PPB.K
...I..
...ICC
......
```

Output: res5.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test5.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
2
Reading config file: test/test5.txt
Config file read successfully
Door side: RIGHT
Door position: 5, 2
Piece count: 3
Board size: 6x6
Running UCS algorithm...
Total nodes visited: 3018
Time taken: 14.877900000000409 ms
Papan awal
....B.
....B.
..PPB.K
...I..
...ICC
......
```

```
6 6
3
......
......
..PPBBK
...I..
...ICC
......
```

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test6.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
2
Reading config file: test/test6.txt
Config file read successfully
Door side: RIGHT
Door position: 5, 2
Piece count: 3
Board size: 6x6
Running UCS algorithm...
Total nodes visited: 1448
Time taken: 10.970800000000054 ms
No solution found.
Simpan ke file? (y/n): y
Nama file : test/res6.txt
Solusi disimpan ke test/res6.txt
```

```
4 4
2
 K
....
.LLA
.P.A
.P..
```

Output: res7.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test7.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
2
Reading config file: test/test7.txt
Config file read successfully
Door side: UP
Door position: 1, 0
Piece count: 2
Board size: 4x4
Running UCS algorithm...
Total nodes visited: 90
Time taken: 1.073099999999613 ms
Papan awal
 K
....
.LLA
.P.A
.P..

Gerakan 1: A-bawah
 K
....
.LL.
.P.A
.P.A

Gerakan 2: L-kanan
 K
....
..LL
.P.A
.P.A

Gerakan 3: P-atas
 K
.P..
.PLL
...A
...A
```

```
10 10
9
.ABBCD....
.A.UCDII..
.PPU.G....K
.H...G....
.HJJJJ....
..........
..........
..........
..........
..........
```

Output: res8.txt

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test8.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
2
Reading config file: test/test8.txt
Config file read successfully
Door side: RIGHT
Door position: 9, 2
Piece count: 9
Board size: 10x10
Running UCS algorithm...
Total nodes visited: 352705
Time taken: 17413.134700000002 ms
Papan awal
.ABBCD....
.A.UCDII..
.PPU.G....K
.H...G....
.HJJJJ....
..........
..........
..........
..........
..........

Gerakan 1: J-kanan
```

```
6 6
8
.ABBCD
.A.UCD
.PPU.GK
..YY.G
...H..
...H..
```

```
PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js
Nama file konfigurasi: test/test9.txt
Pilih algoritma:
| 1: A*    | 2: UCS   | 3: Greedy|
3
Pilih heuristic:
 1: Jumlah piece antara primary piece dan pintu
 2: Jarak primary piece ke pintu
2
Reading config file: test/test9.txt
Config file read successfully
Door side: RIGHT
Door position: 5, 2
Piece count: 8
Board size: 6x6
Running Greedy algorithm...
Total nodes visited: 55058
Time taken: 159.90190000000075 ms
Papan awal
.ABBCD
.A.UCD
.PPU.GK
..YY.G
...H..
...H..
```

6 6 13 .AABB. CCDDEF GHPPEFK GHUVEF GHUVNN .XXYY.	PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js Nama file konfigurasi: test/test10.txt Pilih algoritma: | 1: A* | 2: UCS | 3: Greedy| 3 Pilih heuristic: 1: Jumlah piece antara primary piece dan pintu 2: Jarak primary piece ke pintu 1 Reading config file: test/test10.txt Config file read successfully Door side: RIGHT Door position: 5, 2 Piece count: 13 Board size: 6x6 Running Greedy algorithm... Total nodes visited: 14863 Time taken: 26.95899999999833 ms Papan awal .AABB. CCDDEF GHPPEFK GHUVEF GHUVNN .XXYY.
6 6 11 .B.YYY OB.CD. OPPCDMK IGGGDM I.V..N ..VUUN	PS E:\Project\Kuliah\IF2211 Strategi Algoritma\Tucil3_13523063> node src/Main.js Nama file konfigurasi: test/test11.txt Pilih algoritma: | 1: A* | 2: UCS | 3: Greedy| 3 Pilih heuristic: 1: Jumlah piece antara primary piece dan pintu 2: Jarak primary piece ke pintu 2 Reading config file: test/test11.txt Config file read successfully Door side: RIGHT Door position: 5, 2 Piece count: 11 Board size: 6x6 Running Greedy algorithm... Total nodes visited: 68226 Time taken: 122.80040000000008 ms Papan awal .B.YYY OB.CD. OPPCDMK IGGGDM I.V..N ..VUUN

6 6 12 ABB.C. ADE.CF ADEPPFK ZZZY.F ..WYNN TTWLL.	

6. Lampiran

Link spesifikasi tugas: Spesifikasi Tugas Kecil 3 Stima 2024/2025

Link repository: https://github.com/rizalkhairan/Tucil3_13523063

Checklist pengerjaan tugas:

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	v	
2. Program berhasil dijalankan	v	
3. Solusi yang diberikan program benar dan mematuhi aturan permainan	v	
4. Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	v	
5. [Bonus] Implementasi algoritma pathfinding alternatif		v
6. [Bonus] Implementasi 2 atau lebih heuristik alternatif		v
7. [Bonus] Program memiliki GUI		v
8. Program dan laporan dibuat (kelompok) sendiri	v	

7. Referensi

[1] Russell, S.J.; Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall.