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

Buege Mahara Putra
13523037

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.

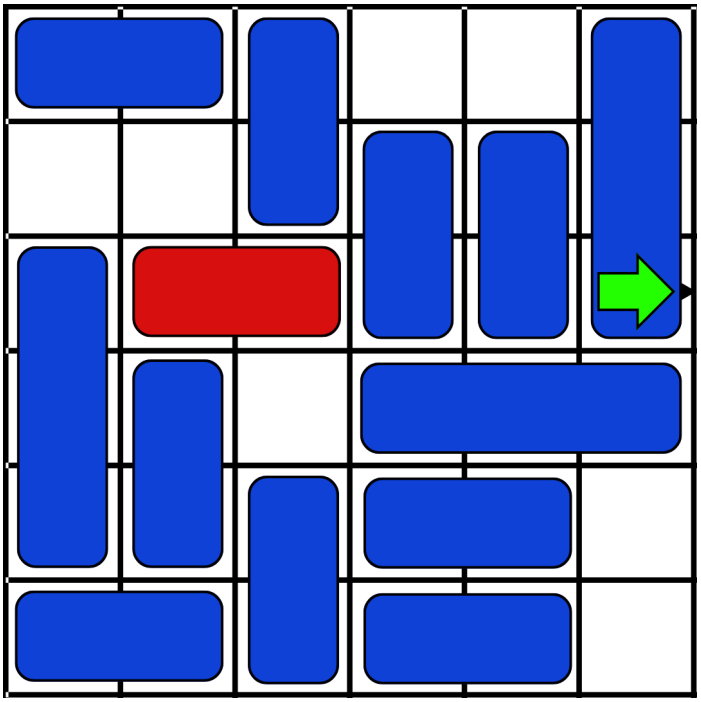
3. **Primary Piece –** *Primary piece* adalah kendaraan utama yang harus dikeluarkan dari *papan* (biasanya berwarna merah). Hanya boleh terdapat satu primary piece.

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.

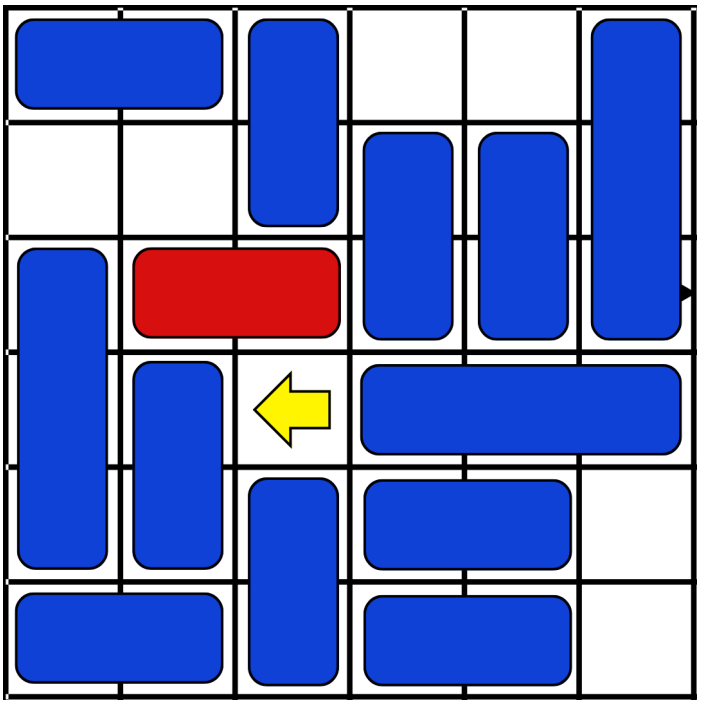
Ilustrasi kasus :

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

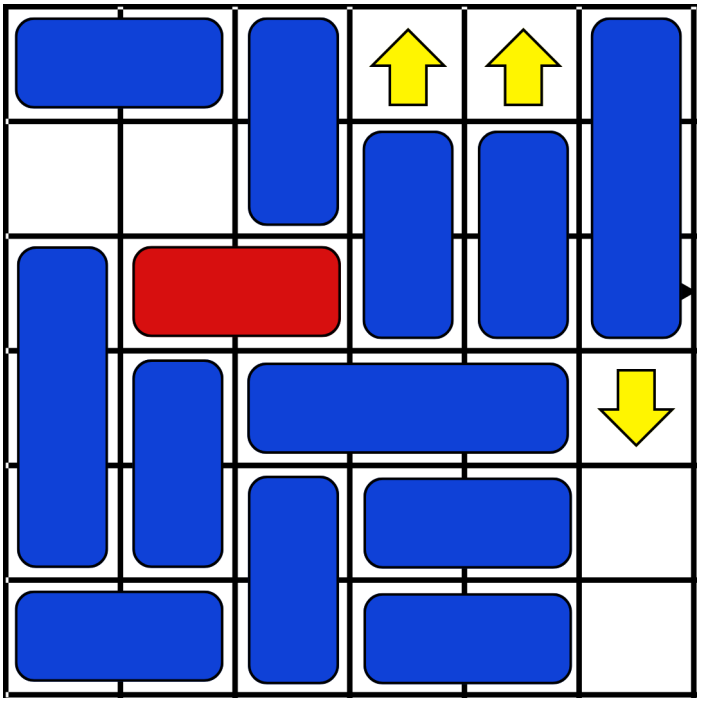


Gambar 2. Awal Permainan Game Rush Hour

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

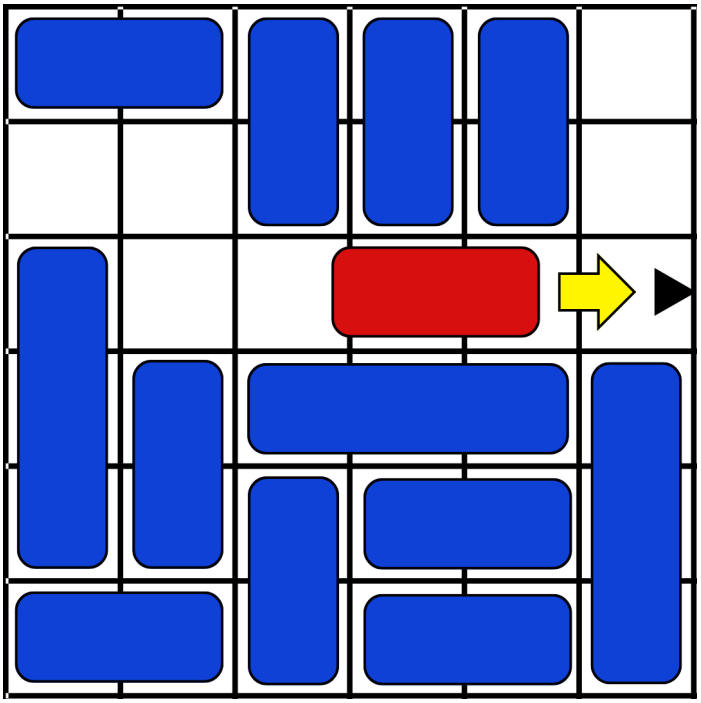


Gambar 3. Gerakan Pertama Game Rush Hour



Gambar 4. Gerakan Kedua Game Rush Hour

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



Gambar 5. Pemain Menyelesaikan Permainan

Bab 2
Algoritma yang Digunakan

2.1. Universal Cost Search (UCS)

Universal Cost Search (UCS) adalah algoritma pathfinding uninformed, yang artinya algoritma ini tidak mengetahui apapun tentang state goal yang ingin dicapai, selain dari masalah yang ingin diselesaikan itu sendiri. UCS memiliki prinsip yang mirip seperti Breadth-First Search. Letak perbedaannya ada pada urutan akses ekspansi simpul. Pada BFS, simpul diakses secara terurut sesuai dengan urutan pada simpul parent-nya, sedangkan pada UCS, simpul diakses terurut berdasarkan cost sisi graf. Cost pada sisi-sisi graf dinyatakan dengan fungsi g(n), sehingga fungsi evaluasi untuk UCS adalah f(n) = g(n). Karena perbedaan tersebut, urutan simpul yang dibangkitkan dan path yang dihasilkan oleh UCS tidak sama persis seperti BFS.

Pada program ini, cost yang digunakan adalah jarak perpindahan suatu piece. Piece yang berpindah jauh dalam satu gerakan memiliki cost yang rendah, dan berlaku pula sebaliknya. Pemilihan cost seperti ini mengarahkan UCS untuk memilih path yang memiliki perpindahan lebih banyak dalam gerakan yang sedikit.

$$g(n) = currentCost + 1 + maxMove - |move|$$

currentCost = cost yang dimiliki parent node
maxMove = perpindahan paling besar yang bisa dilakukan oleh suatu piece
move = perpindahan absolut dari suatu piece

2.2. Greedy Best-First Search (GBFS)

Greedy Best-First Search adalah algoritma pathfinding informed, yang artinya algoritma ini mengetahui tujuan state yang ingin dicapainya menggunakan heuristik. Urutan pengaksesan node dalam GBFS mirip seperti UCS, hanya saja fungsi evaluasi untuk GBFS adalah f(n) = h(n), di mana h(n) adalah heuristik.

Pada program ini, heuristik yang digunakan adalah jumlah piece yang menghalangi primary piece untuk mencapai exit cell dalam satu garis lurus yang ditarik dari primary piece ke exit cell.

$$h(n) = \sum blockingVehicle$$

Secara teoretis, algoritma GBFS tidak menjamin solusi optimal untuk pemecahan masalah ini, karena GBFS hanya mengandalkan heuristik. Jika heuristik salah, algoritma bisa terjebak di maksimum atau minimum lokal.

2.3. A* Search

A* search merupakan salah satu algoritma informed search. A* menggabungkan cost seperti pada UCS dan heuristik seperti pada GBFS. Fungsi evaluasi untuk A* adalah f(n) = g(n) + h(n). Karena A* menggabungkan cost dan heuristik, path yang ditelusuri oleh A* bisa terarah untuk menghindari mengekspansi path yang mahal.

Heuristik h(n) dikatakan admissible jika untuk setiap simpul, h(n) < h*(n), di mana h(n) merupakan cost sebenarnya untuk mencapai goal state dari n. Heuristik yang admissible tidak pernah memperkirakan cost secara berlebihan, yang artinya heuristik ini bersifat optimis. Terdapat teorema jika h(n) admissible, A* menggunakan tree search menghasilkan path yang optimal.

Pada program ini, heuristik yang digunakan admissible, karena heuristik ini menghitung minimal jumlah piece yang perlu dipindahkan agar primary piece bisa keluar. Heuristik ini pada intinya mencari lower bound dari cost yang dibutuhkan, yang pasti selalu lebih rendah dari cost sebenarnya.

Fungsi evaluasi untuk A* search adalah sebagai berikut.

$$f(n) = g(n) + h(n) = \text{currentCost} + 1 + \text{maxMove} - |\text{move}| + \sum \text{blockingVehicle}$$

Secara teoretis, A* lebih efisien dari UCS. Pada kasus terburuk h(n) = 0, A* memiliki performa yang sama dengan UCS, dan pada kasus h(n) ideal, A* jauh lebih efisien dari UCS karena A* memprioritaskan path yang menjanjikan berdasarkan cost sejauh ini dan perkiraan menuju goal.

Bab 3
Source Code Program

Program ini dibuat menggunakan bahasa C++.

3.1. main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <chrono>
#include "game/BoardState.hpp"
#include "algorithms/UCS.hpp"
#include "algorithms/GBFS.hpp"
#include "algorithms/AStar.hpp"
#include "utils/output.hpp"
#include "utils/txt_parser.hpp"

using namespace std;
using namespace std::chrono;

int main(int argc, char* argv[]) {
    if (argc != 2) {
        cout << "Usage: " << argv[0] << " <config_file>" << endl;
        return 1;
    }

    BoardState initial;
    try {
        initial = parseConfig(argv[1]);
        initSolutionFile(argv[1]);
        outputBoth("\n");
    } catch (const exception& e) {
        cout << e.what() << endl << endl;
        return 1;
    }

    outputBoth("Choose algorithm:\n");
    outputBoth("1. UCS (Uniform Cost Search)\n");
    outputBoth("2. GBFS (Greedy Best-First Search)\n");
```

```
    outputBoth("3. A* Search\n");

    int choice;
    do {
        cout << "Choice: ";
        cin >> choice;
        solutionFile << "Choice: " << choice << "\n";
        if (choice < 1 || choice > 3) {
            cout << "Invalid choice. Please try again." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    } while (choice < 1 || choice > 3);

    outputBoth("\nInitial board:\n");
    printColoredBoard(initial);
    outputBoth("\n");

    system_clock::time_point start, end;
    vector<pair<char, int>> solution;
    int nodesVisited;

    UCS ucs;
    GBFS gbfs;
    AStar astar;

    if (choice == 1) {
        start = high_resolution_clock::now();
        solution = ucs.solve(initial);
        end = high_resolution_clock::now();
        nodesVisited = ucs.getNodesVisited();
    } else if (choice == 2) {
        start = high_resolution_clock::now();
        solution = gbfs.solve(initial);
        end = high_resolution_clock::now();
        nodesVisited = gbfs.getNodesVisited();
    } else {
        start = high_resolution_clock::now();
```

```
        solution = astar.solve(initial);
        end = high_resolution_clock::now();
        nodesVisited = astar.getNodesVisited();
    }

    if (solution.empty()) {
        outputBoth("No solution found :(\n");
        return 0;
    }

    // Replay solution
    BoardState current = initial;
    auto vehicles = current.getVehicles();

    for (size_t i = 0; i < solution.size(); i++) {
        const auto& [ID, move] = solution[i];
        outputBoth("Move " + to_string(i+1) + ": " + ID + "-" +
                   getMoveDirection(vehicles, ID, move) + "\n");

        if (i == solution.size() - 1) vehicles.erase(playerID);
        else vehicles[ID] = vehicles[ID].moved(move, current.getBoard());
        current = BoardState(vehicles);

        printColoredBoard(current, ID);
        outputBoth("\n");
    }

    auto duration = duration_cast<milliseconds>(end - start);
    outputBoth("Nodes visited: " + to_string(nodesVisited) + "\n");
    outputBoth("Execution time: " + to_string(duration.count()) + "ms\n");

    solutionFile.close();
    return 0;
}
```

3.2. game/BoardConfig.hpp

```
#ifndef __BOARD_CONFIG_HPP__
```

```
#define __BOARD_CONFIG_HPP__

#include <utility>
using namespace std;

class BoardConfig {
public:
    static pair<int, int> dimension;  // rows, cols
    static pair<int, int> exitCoord;  // row, col
};

#endif
```

3.3. game/BoardConfig.cpp

```
#include "BoardConfig.hpp"

pair<int, int> BoardConfig::dimension;
pair<int, int> BoardConfig::exitCoord;
```

3.4. game/BoardState.hpp

```
#ifndef __BOARD_STATE_HPP__
#define __BOARD_STATE_HPP__

#include "BoardConfig.hpp"
#include "Vehicle.hpp"
#include <unordered_map>
#include <vector>
#include <string>

using namespace std;

class BoardState : public BoardConfig {
    private:
        unordered_map<char, Vehicle> vehicles;
        vector<vector<char>> board;
        vector<string> debugBoard;
```

```
    public:
        BoardState() = default;
        BoardState(unordered_map<char, Vehicle> vehicles);

        const vector<vector<char>>& getBoard() const { return board; }
        const unordered_map<char, Vehicle>& getVehicles() const { return
vehicles; }

        bool isGoal();

        bool operator==(const BoardState& other) const { return vehicles ==
other.vehicles; }
};

#endif
```

3.5. game/BoardState.cpp

```
#include "BoardState.hpp"

BoardState::BoardState(unordered_map<char, Vehicle> vehicles) :
vehicles(vehicles) {
    board = vector<vector<char>>(dimension.first,
vector<char>(dimension.second, '.'));
    for (const auto& [id, vehicle] : vehicles) {
        for (int i = 0; i < vehicle.getLength(); ++i) {
            int r = vehicle.getRow() + (vehicle.getOrientation() == VERTICAL ?
i : 0);
            int c = vehicle.getCol() + (vehicle.getOrientation() == HORIZONTAL
? i : 0);
            board[r][c] = id;
        }
    }

    debugBoard = vector<string>(dimension.first);
    for (int i = 0; i < dimension.first; ++i) {
```

```
        debugBoard[i] = string(board[i].begin(), board[i].end());
    }
}

bool BoardState::isGoal() {
    const auto& player = vehicles.at(playerID);
    if (player.getOrientation() == HORIZONTAL) {
        // right
        if (player.getCol() + player.getLength() ==
BoardState::dimension.second &&
            BoardState::exitCoord.second == BoardState::dimension.second &&
            player.getRow() == BoardState::exitCoord.first) {
            return true;
        }
        // left
        if (player.getCol() == 0 &&
            BoardState::exitCoord.second == -1 &&
            player.getRow() == BoardState::exitCoord.first) {
            return true;
        }
    } else {
        // bottom
        if (player.getRow() + player.getLength() == BoardState::dimension.first
&&
            BoardState::exitCoord.first == BoardState::dimension.first &&
            player.getCol() == BoardState::exitCoord.second) {
            return true;
        }
        // top
        if (player.getRow() == 0 &&
            BoardState::exitCoord.first == -1 &&
            player.getCol() == BoardState::exitCoord.second) {
            return true;
        }
    }

    return false;
}
```

3.6. game/Vehicle.hpp

```
#ifndef __VEHICLE_HPP__
#define __VEHICLE_HPP__

#include <string>
#include <vector>
#include <stdexcept>
#include "BoardConfig.hpp"

using namespace std;

enum Orientation {
    HORIZONTAL,
    VERTICAL
};

const char playerID = 'P';
const char exitID = 'K';

class Vehicle {
    private:
        char ID;
        int row, col; // top left coordinate
        int length;
        Orientation orientation;

    public:
        Vehicle() : ID('\0'), row(0), col(0), length(0),
orientation(HORIZONTAL) {};

        Vehicle(const char ID, int row, int col, int length, Orientation
orientation) :
            ID(ID), row(row), col(col), length(length),
orientation(orientation) {};

        char getID() const { return ID; }
        int getRow() const { return row; }
```

```
        int getCol() const { return col; }
        int getLength() const { return length; }
        Orientation getOrientation() const { return orientation; }

        Vehicle moved(int move, const vector<vector<char>>& board) const;

        bool operator==(const Vehicle& other) const;
};

#endif
```

3.7. game/Vehicle.cpp

```
#include "Vehicle.hpp"

Vehicle Vehicle::moved(int move, const vector<vector<char>>& board) const {
    if (orientation == HORIZONTAL) {
        int newCol = col + move;
        if (newCol < 0 || newCol + length - 1 >= BoardConfig::dimension.second)
{
            return *this;
        }

        // check collision
        if (move < 0) {
            for (int c = col - 1; c >= col + move; c--) {
                char cell = board[row][c];
                if (cell != '.') return *this;
            }
        } else {
            for (int c = col + length; c <= col + length + move - 1; c++) {
                char cell = board[row][c];
                if (cell != '.') return *this;
            }
        }

        return Vehicle(ID, row, newCol, length, orientation);
    } else {
```

```
        int newRow = row + move;
        if (newRow < 0 || newRow + length - 1 >= BoardConfig::dimension.first)
{
            return *this;
        }

        // check collision
        if (move < 0) {
            for (int r = row - 1; r >= row + move; r--) {
                char cell = board[r][col];
                if (cell != '.') return *this;
            }
        } else {
            for (int r = row + length; r <= row + length + move - 1; r++) {
                char cell = board[r][col];
                if (cell != '.') return *this;
            }
        }

        return Vehicle(ID, newRow, col, length, orientation);
    }
}

bool Vehicle::operator==(const Vehicle& other) const {
    return ID == other.ID &&
           row == other.row &&
           col == other.col &&
           length == other.length &&
           orientation == other.orientation;
}
```

3.8. algorithms/Search.hpp

```
#ifndef __SEARCH_HPP__
#define __SEARCH_HPP__

#include <algorithm>
#include <queue>
```

```cpp
#include <unordered_set>
#include <vector>
#include "../game/BoardState.hpp"
#include "../utils/hash.hpp"

template<typename NodeType>
class Search {
protected:
    int nodesVisited = 0;
    vector<NodeType*> allNodes;
    priority_queue<NodeType*, vector<NodeType*>, typename NodeType::Compare>
queue;
    unordered_set<BoardState> explored;

    vector<pair<char, int>> reconstructPath(NodeType* goal) {
        vector<pair<char, int>> path;
        NodeType* n = goal;
        while (n->parent != nullptr) {
            path.push_back(n->move);
            n = n->parent;
        }
        reverse(path.begin(), path.end());
        return path;
    }

    void cleanupQueue(priority_queue<NodeType*, vector<NodeType*>, typename
NodeType::Compare>& queue) {
        while (!queue.empty()) {
            delete queue.top();
            queue.pop();
        }
    }

    virtual vector<NodeType*> getNextStates(NodeType* current) = 0;
    virtual NodeType* createStartNode(const BoardState& initial) = 0;

public:
    int getNodesVisited() const { return nodesVisited; }
```

```
vector<pair<char, int>> solve(const BoardState& initial) {
    NodeType* start = createStartNode(initial);
    queue.push(start);
    allNodes.push_back(start);

    while (!queue.empty()) {
        nodesVisited++;
        NodeType* current = queue.top();
        queue.pop();

        if (explored.find(current->state) != explored.end()) {
            continue;
        }

        explored.insert(current->state);

        if (current->state.isGoal()) {
            return reconstructPath(current);
        }

        vector<NodeType*> children = getNextStates(current);
        for (auto nextState : children) {
            if (explored.find(nextState->state) != explored.end()) {
                delete nextState;
                continue;
            }
            queue.push(nextState);
            allNodes.push_back(nextState);
        }
    }
    return {};
}

virtual ~Search() {
    for (auto node : allNodes) {
        delete node;
    }
```

```
    }
};

#endif
```

3.9. algorithms/SearchNodes.hpp

```
#ifndef __SEARCH_UTILS_HPP__
#define __SEARCH_UTILS_HPP__

#include "../game/BoardState.hpp"
#include "../utils/heuristic.hpp"

template<typename T>
struct SearchNode {
    BoardState state;
    T* parent;
    pair<char, int> move;

    SearchNode(BoardState state, T* parent, pair<char, int> move)
        : state(state), parent(parent), move(move) {}
    virtual ~SearchNode() = default;
};

struct UCSNode : public SearchNode<UCSNode> {
    int cost;

    UCSNode(BoardState state, int cost, UCSNode* parent, pair<char, int> move)
        : SearchNode<UCSNode>(state, parent, move), cost(cost) {}

    struct Compare {
        bool operator()(const UCSNode* a, const UCSNode* b) const {
            return a->cost > b->cost;
        }
    };
};
```

```
struct GBFSNode : public SearchNode<GBFSNode> {
    int heuristic;

    GBFSNode(BoardState state, int heuristic, GBFSNode* parent, pair<char, int>
move)
        : SearchNode<GBFSNode>(state, parent, move), heuristic(heuristic) {}

    struct Compare {
        bool operator()(const GBFSNode* a, const GBFSNode* b) const {
            return a->heuristic > b->heuristic;
        }
    };
};

struct AStarNode : public SearchNode<AStarNode> {
    int cost;
    int heuristic;

    AStarNode(BoardState state, int cost, int heuristic, AStarNode* parent,
pair<char, int> move)
        : SearchNode<AStarNode>(state, parent, move), cost(cost),
heuristic(heuristic) {}

    int getF() const { return cost + heuristic; }

    struct Compare {
        bool operator()(const AStarNode* a, const AStarNode* b) const {
            return a->getF() > b->getF();
        }
    };
};

#endif
```

3.10. algorithms/UCS.hpp

```
#ifndef __UCS_HPP__
#define __UCS_HPP__

#include "Search.hpp"
#include "SearchNodes.hpp"

class UCS : public Search<UCSNode> {
protected:
    vector<UCSNode*> getNextStates(UCSNode* current) override;
    UCSNode* createStartNode(const BoardState& initial) override;
};

#endif
```

3.11. algorithms/UCS.cpp

```
#include "UCS.hpp"

vector<UCSNode*> UCS::getNextStates(UCSNode* current) {
    vector<UCSNode*> nextStates;

    const auto& vehicles = current->state.getVehicles();
    for (const auto& [id, vehicle] : vehicles) {
        int maxMove = vehicle.getOrientation() == HORIZONTAL ?
            BoardState::dimension.second - vehicle.getLength() :
            BoardState::dimension.first - vehicle.getLength();

        for (int move = -maxMove; move <= maxMove; ++move) {
            if (move == 0) continue;

            Vehicle moved = vehicle.moved(move, current->state.getBoard());
            if (moved == vehicle) continue;

            auto newVehicles = vehicles;
            newVehicles[id] = moved;
```

```
            UCSNode* child = new UCSNode(
                BoardState(newVehicles),
                current->cost + calculateNextCost(maxMove, move),
                current,
                {id, move}
            );

            nextStates.push_back(child);
        }
    }

    return nextStates;
}

UCSNode* UCS::createStartNode(const BoardState& initial) {
    return new UCSNode(initial, 0, nullptr, {'\0', 0});
}
```

3.12. algorithms/GBFS.hpp

```
#ifndef __GBFS_HPP__
#define __GBFS_HPP__

#include "Search.hpp"
#include "SearchNodes.hpp"

class GBFS : public Search<GBFSNode> {
protected:
    vector<GBFSNode*> getNextStates(GBFSNode* current) override;
    GBFSNode* createStartNode(const BoardState& initial) override;
};

#endif
```

3.13. algorithms/GBFS.cpp

```cpp
#include "GBFS.hpp"

vector<GBFSNode*> GBFS::getNextStates(GBFSNode* current) {
    vector<GBFSNode*> nextStates;

    const auto& vehicles = current->state.getVehicles();
    for (const auto& [id, vehicle] : vehicles) {
        int maxMove = vehicle.getOrientation() == HORIZONTAL ?
            BoardState::dimension.second - vehicle.getLength() :
            BoardState::dimension.first - vehicle.getLength();

        for (int move = -maxMove; move <= maxMove; ++move) {
            if (move == 0) continue;

            Vehicle moved = vehicle.moved(move, current->state.getBoard());
            if (moved == vehicle) continue;

            auto newVehicles = vehicles;
            newVehicles[id] = moved;
            BoardState nextState(newVehicles);

            GBFSNode* child = new GBFSNode(
                nextState,
                calculateBlockingHeuristic(nextState),
                current,
                {id, move}
            );

            nextStates.push_back(child);
        }
    }

    return nextStates;
}

GBFSNode* GBFS::createStartNode(const BoardState& initial) {
```

```
    return new GBFSNode(initial, calculateBlockingHeuristic(initial), nullptr,
{'\0', 0});
}
```

3.14. algorithms/AStar.hpp

```
#ifndef __A_STAR_HPP__
#define __A_STAR_HPP__

#include "Search.hpp"
#include "SearchNodes.hpp"

class AStar : public Search<AStarNode> {
protected:
    vector<AStarNode*> getNextStates(AStarNode* current) override;
    AStarNode* createStartNode(const BoardState& initial) override;
};

#endif
```

3.15. algorithms/AStar.cpp

```
#include "AStar.hpp"

vector<AStarNode*> AStar::getNextStates(AStarNode* current) {
    vector<AStarNode*> nextStates;

    const auto& vehicles = current->state.getVehicles();
    for (const auto& [id, vehicle] : vehicles) {
        int maxMove = vehicle.getOrientation() == HORIZONTAL ?
            BoardState::dimension.second - vehicle.getLength() :
            BoardState::dimension.first - vehicle.getLength();

        for (int move = -maxMove; move <= maxMove; ++move) {
            if (move == 0) continue;
```

```
            Vehicle moved = vehicle.moved(move, current->state.getBoard());
            if (moved == vehicle) continue;

            auto newVehicles = vehicles;
            newVehicles[id] = moved;
            BoardState nextState(newVehicles);

            AStarNode* child = new AStarNode(
                nextState,
                current->cost + calculateNextCost(maxMove, move),
                calculateBlockingHeuristic(nextState),
                current,
                {id, move}
            );

            nextStates.push_back(child);
        }
    }

    return nextStates;
}

AStarNode* AStar::createStartNode(const BoardState& initial) {
    return new AStarNode(initial, 0, calculateBlockingHeuristic(initial),
nullptr, {'\0', 0});
}
```

3.16. utils/hash.hpp

```
#ifndef __HASH_HPP__
#define __HASH_HPP__

#include <functional>
#include "../game/BoardState.hpp"

namespace std {
    template<>
    struct hash<Vehicle> {
```

```
        size_t operator()(const Vehicle& v) const {
            const size_t prime = 0x9e3779cd;
            size_t seed = hash<char>()(v.getID());
            seed ^= hash<int>()(v.getRow()) + prime + (seed << 6) + (seed >>
2);
            seed ^= hash<int>()(v.getCol()) + prime + (seed << 6) + (seed >>
2);
            seed ^= hash<int>()(v.getLength()) + prime + (seed << 6) + (seed >>
2);
            seed ^= hash<bool>()(v.getOrientation() == HORIZONTAL) + prime +
(seed << 6) + (seed >> 2);
            return seed;
        }
    };

    template<>
    struct hash<BoardState> {
        size_t operator()(const BoardState& state) const {
            const size_t prime = 0x9e3779cd;
            size_t seed = 0;
            for (const auto& [id, v] : state.getVehicles()) {
                size_t vehicle_hash = hash<Vehicle>()(v);
                seed ^= vehicle_hash + prime + (seed << 6) + (seed >> 2);
            }
            return seed;
        }
    };
}

#endif
```

3.17. utils/heuristic.hpp

```
#ifndef __HEURISTIC_HPP__
#define __HEURISTIC_HPP__

#include "../game/BoardState.hpp"
#include <unordered_set>
```

```
int calculateNextCost(int maxMove, int move);
int calculateBlockingHeuristic(const BoardState& state);

#endif
```

3.18. utils/heuristic.cpp

```
#include "heuristic.hpp"

int calculateNextCost(int maxMove, int move) {
    return 1 + maxMove - abs(move);
}


int calculateBlockingHeuristic(const BoardState& state) {
    const auto& vehicles = state.getVehicles();
    const Vehicle& player = vehicles.at(playerID);
    unordered_set<char> blocking;

    const auto& board = state.getBoard();

    if (player.getOrientation() == HORIZONTAL) {
        if (BoardState::exitCoord.second == BoardState::dimension.second) {
            for (int c = player.getCol() + player.getLength(); c <
BoardState::dimension.second; c++) {
                if (board[player.getRow()][c] != '.') {
                    blocking.insert(board[player.getRow()][c]);
                }
            }
        }
        else if (BoardState::exitCoord.second == -1) {
            for (int c = 0; c < player.getCol(); c++) {
                if (board[player.getRow()][c] != '.') {
                    blocking.insert(board[player.getRow()][c]);
                }
            }
        }
```

```
    } else {
        if (BoardState::exitCoord.first == BoardState::dimension.first) {
            for (int r = player.getRow() + player.getLength(); r <
BoardState::dimension.first; r++) {
                if (board[r][player.getCol()] != '.') {
                    blocking.insert(board[r][player.getCol()]);
                }
            }
        }
        else if (BoardState::exitCoord.first == -1) {
            for (int r = 0; r < player.getRow(); r++) {
                if (board[r][player.getCol()] != '.') {
                    blocking.insert(board[r][player.getCol()]);
                }
            }
        }
    }

    return blocking.size();
}
```

3.19. utils/output.hpp

```
#ifndef __OUTPUT_HPP__
#define __OUTPUT_HPP__

#include "../game/BoardState.hpp"
#include <string>
#include <fstream>

using namespace std;

void printColoredBoard(const BoardState& state, char movedPiece = '\0');
string getMoveDirection(const unordered_map<char, Vehicle>& vehicles, char ID,
int move);
extern ofstream solutionFile;
void initSolutionFile(const string& configPath);
```

```
void outputBoth(const string& text);

#endif
```

3.20. utils/output.cpp

```
#include "output.hpp"
#include <iostream>
#include <filesystem>

#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define RESET "\033[0m"

void printColoredBoard(const BoardState& state, char movedVehicleID) {
    auto [rows, cols] = BoardConfig::dimension;
    auto [exitRow, exitCol] = BoardConfig::exitCoord;
    auto board = state.getBoard();

    bool isTop = exitRow == -1;
    bool isBottom = exitRow == rows;
    bool isLeft = exitCol == -1;
    bool isRight = exitCol == cols;

    if (isTop) {
        cout << endl;
        solutionFile << endl;
    }

    for (int i = 0; i < cols + 2; i++) {
        if (isTop && i == exitCol + 1) {
            cout << GREEN << 'K' << RESET << "  ";
            solutionFile << 'K' << "  ";
        } else {
            outputBoth(" " "  ");
        }
    }
```

```
    outputBoth("\n");

    for (int i = 0; i < rows; i++) {
        if (isLeft && i == exitRow) {
            cout << GREEN << 'K' << RESET << "  ";
            solutionFile << 'K' << "  ";
        } else {
            outputBoth(" " "  ");
        }

        for (int j = 0; j < cols; j++) {
            char cell = board[i][j];
            if (cell == 'P') {
                cout << RED << cell << RESET << "  ";
                solutionFile << cell << "  ";
            } else if (cell == movedVehicleID) {
                cout << YELLOW << cell << RESET << "  ";
                solutionFile << cell << "  ";
            } else {
                outputBoth(string(1, cell) + "  ");
            }
        }

        if (isRight && i == exitRow) {
            cout << GREEN << 'K' << RESET << "  \n";
            solutionFile << 'K' << "  \n";
        } else {
            outputBoth(" \n");
        }
    }

    for (int i = 0; i < cols + 2; i++) {
        if (isBottom && i == exitCol + 1) {
            cout << GREEN << 'K' << RESET << "  ";
            solutionFile << 'K' << "  ";
        } else {
            outputBoth(" " "  ");
        }
```

```
    }
    outputBoth("\n");
}

string getMoveDirection(const unordered_map<char, Vehicle>& vehicles, char ID,
int move) {
    auto direction = vehicles.at(ID).getOrientation();
    if (direction == HORIZONTAL) {
        if (move < 0) return "left";
        else return "right";
    } else {
        if (move < 0) return "up";
        else return "down";
    }
}

ofstream solutionFile;

void initSolutionFile(const string& configPath) {
    filesystem::create_directories("test/solutions");
    string filename = filesystem::path(configPath).filename().string();
    solutionFile.open("test/solutions/" + filename);
}

void outputBoth(const string& text) {
    cout << text;
    if (solutionFile.is_open()) {
        solutionFile << text;
    }
}
```

3.21. utils/txt_parser.hpp

```
#ifndef __TXT_PARSER_HPP__
#define __TXT_PARSER_HPP__
```

```
#include "../game/BoardState.hpp"
#include "../game/BoardConfig.hpp"
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdexcept>
#include <string>

BoardState parseConfig(const string& filename);

#endif
```

3.22. utils/txt_parser.cpp

```
#include "txt_parser.hpp"

BoardState parseConfig(const string& filename) {
    cout << "\nLoading " << filename << " ..." << endl;

    // open file
    ifstream configFile(filename);
    if (!configFile.is_open()) throw runtime_error("Cannot open " + filename);

    // take rows and cols
    int rows, cols;
    configFile >> rows >> cols;

    if (rows <= 0 || cols <= 0 || rows * cols < 2) {
        throw runtime_error("Invalid board dimension.\nBoard dimension must be
at least 2x1 or 1x2.");
    }

    // take num of pieces
    int numPieces;
    configFile >> numPieces;

    vector<string> board;
```

```
    string line;
    while (getline(configFile, line)) {
        if (line == "" || line == "\r") continue;
        board.emplace_back(line);
    }

    unordered_map<char, vector<pair<int, int>>> positions;
    pair<int, int> exitCoord(-2, -2);
    pair<int, int> topLeft, bottomRight;
    bool hasTopLeft = false;

    // take positions data
    for (size_t r = 0; r < board.size(); ++r) {
        for (size_t c = 0; c < board[r].size(); ++c) {
            char ch = board[r][c];

            if (ch == 'K') {
                exitCoord = {r, c};
                continue;
            }

            if (!('A' <= ch && ch <= 'Z') && ch != '.') {
                continue;
            }

            if (hasTopLeft) {
                bottomRight = {static_cast<int>(r), static_cast<int>(c)};
            } else {
                topLeft = {static_cast<int>(r), static_cast<int>(c)};
                hasTopLeft = true;
            }

            if (ch != '.') {
                positions[ch].emplace_back(r, c);
            }
        }
    }
```

```
    if (bottomRight.first - topLeft.first + 1 != rows || bottomRight.second -
topLeft.second + 1 != cols) {
        throw runtime_error("The board dimension does not match the stated
number.");
    }

    if (exitCoord.first == -2 && exitCoord.second == -2) {
        throw runtime_error("There is no exit on the board.");
    }

    // calculate offset caused by exit (if on top or left)
    pair<int, int> exitOffset(0, 0);

    if (exitCoord.first == 0 && static_cast<size_t>(rows) < board.size()) { //
top
        exitOffset = {-1, 0};
        exitCoord.first -= 1;
    } else if (exitCoord.second == 0 && static_cast<size_t>(rows) ==
board.size()) { // left
        exitOffset = {0, -1};
        exitCoord.second -= 1;
    }

    // process position data into vehicles
    unordered_map<char, Vehicle> vehicles;

    for (const auto& [id, cells] : positions) {
        int r0 = cells[0].first, c0 = cells[0].second;
        int length = cells.size();
        Orientation orientation = HORIZONTAL;
        if (cells[1].first != r0) orientation = VERTICAL;

        vehicles[id] = Vehicle(id, r0 + exitOffset.first, c0 +
exitOffset.second, length, orientation);
    }

    if (static_cast<size_t>(numPieces) != vehicles.size() - 1) {
        throw runtime_error(string("The number of pieces on the board does not
```

```
match the stated number. Found ") + to_string(vehicles.size() - 1) + "
pieces.");
    }

    if (vehicles.find('P') == vehicles.end()) {
        throw runtime_error("There is no primary piece on the board.");
    }

    if (exitCoord.first != vehicles[playerID].getRow() && exitCoord.second !=
vehicles[playerID].getCol()) {
        throw runtime_error("Impossible board. Primary piece is not aligned
with exit cell.");
    }

    // set config and return
    BoardConfig::dimension = {rows, cols};
    BoardConfig::exitCoord = exitCoord;

    return BoardState(vehicles);
}
```

Bab 4
Tangkapan Layar

4.1. Universal Cost Search (UCS)

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/1.txt

Loading ./test/problems/1.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 1

Initial board:

  A  B  B  .  H  .
  A  C  D  .  H  I
  A  C  D  P  P  I  K
  E  E  E  M  .  I
  .  .  F  M  J  J
  G  G  F  L  L  .


Move 1: L-right

  A  B  B  .  H  .
  A  C  D  .  H  I
  A  C  D  P  P  I  K
  E  E  E  M  .  I
  .  .  F  M  J  J
  G  G  F  .  L  L


Move 2: I-up

  A  B  B  .  H  I
  A  C  D  .  H  I
  A  C  D  P  P  I  K
  E  E  E  M  .  .
  .  .  F  M  J  J
```

```
Move 50: I-down

  B  B  D  M  H  .
  A  .  D  M  H  .
  A  .  P  P  .  .  K
  A  C  E  E  E  I
  .  C  F  J  J  I
  G  G  F  L  L  I


Move 51: P-right

  B  B  D  M  H  .
  A  .  D  M  H  .
  A  .  .  .  .  .  K
  A  C  E  E  E  I
  .  C  F  J  J  I
  G  G  F  L  L  I


Nodes visited: 11638
Execution time: 623ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/2.txt

Loading ./test/problems/2.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 1

Initial board:

      K
  .  A  A  B  D  D
  .  .  .  B  C  .
  G  G  G  B  C  F
  I  H  H  E  E  F
  I  .  P  .  .  L
  .  .  P  J  J  L


Move 1: I-down

      K
  .  A  A  B  D  D
  .  .  .  B  C  .
  G  G  G  B  C  F
  .  H  H  E  E  F
  I  .  P  .  .  L
  I  .  P  J  J  L


Move 2: A-left

      K
  A  A  .  B  D  D
```

```
Move 35: A-left

      K
  A  A  .  D  D  F
  I  .  .  .  .  F
  I  .  .  G  G  G
  H  H  .  B  E  E
  .  .  P  B  C  L
  J  J  P  B  C  L


Move 36: P-up

      K
  A  A  .  D  D  F
  I  .  .  .  .  F
  I  .  .  G  G  G
  H  H  .  B  E  E
  .  .  .  B  C  L
  J  J  .  B  C  L


Nodes visited: 2020
Execution time: 106ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/3.txt

Loading ./test/problems/3.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 1

Initial board:

  A B B B . .
  A C C C G .
  P P E F G H K
  D D E F G H
  . . . J I I
  . L L J . .


Move 1: L-left

  A B B B . .
  A C C C G .
  P P E F G H K
  D D E F G H
  . . . J I I
  L L . J . .


Move 2: E-down

  A B B B . .
  A C C C G .
  P P . F G H K
  D D . F G H
  . . E J I I
```

```
Move 44: F-up

  B B B F . .
  C C C F . .
  A P P . . . K
  A D D . G .
  I I E J G H
  L L E J G H


Move 45: P-right

  B B B F . .
  C C C F . .
  A . . . . . K
  A D D . G .
  I I E J G H
  L L E J G H


Nodes visited: 37828
Execution time: 2068ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/4.txt

Loading ./test/problems/4.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 1

Initial board:

   .  .  B  C  C  .
   A  A  B  .  .  .
   D  E  F  G  H  H
K  D  E  F  G  P  P
   .  E  I  I  I  L
   .  .  J  J  J  L


Move 1: C-right

   .  .  B  .  C  C
   A  A  B  .  .  .
   D  E  F  G  H  H
K  D  E  F  G  P  P
   .  E  I  I  I  L
   .  .  J  J  J  L


Move 2: G-up

   .  .  B  G  C  C
   A  A  B  G  .  .
   D  E  F  .  H  H
K  D  E  F  .  P  P
   .  E  I  I  I  L
   .  .  J  J  J  L
```

```
Move 44: F-down

   D  E  B  G  C  C
   D  E  B  G  A  A
   .  E  .  H  H  L
K  .  .  .  P  P  L
   .  .  F  I  I  I
   .  .  F  J  J  J


Move 45: P-left

   D  E  B  G  C  C
   D  E  B  G  A  A
   .  E  .  H  H  L
K  .  .  .  .  .  L
   .  .  F  I  I  I
   .  .  F  J  J  J


Nodes visited: 37795
Execution time: 1887ms
```

4.2. Greedy Best-First Search

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/5.txt

Loading ./test/problems/5.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 2

Initial board:

   A  .  C  D  D  D
   A  .  C  P  .  E
   B  B  C  P  F  E
   .  I  J  J  F  G
   H  I  L  L  L  G
   H  I  M  M  M  .
            K

Move 1: F-up

   A  .  C  D  D  D
   A  .  C  P  F  E
   B  B  C  P  F  E
   .  I  J  J  .  G
   H  I  L  L  L  G
   H  I  M  M  M  .
            K

Move 2: H-up

   A  .  C  D  D  D
   A  .  C  P  F  E
   B  B  C  P  F  E
   H  I  J  J  .  G
   H  I  L  L  L  G
   .  I  M  M  M  .
            K
```

```
Move 51: L-left

   A  D  D  D  F  E
   A  I  C  P  F  E
   H  I  C  P  B  B
   H  I  C  .  J  J
   L  L  L  .  .  G
   M  M  M  .  .  G
            K

Move 52: P-down

   A  D  D  D  F  E
   A  I  C  .  F  E
   H  I  C  .  B  B
   H  I  C  .  J  J
   L  L  L  .  .  G
   M  M  M  .  .  G
            K

Nodes visited: 538
Execution time: 46ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/6.txt

Loading ./test/problems/6.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 2

Initial board:

   A  .  B  .  .  .
   A  .  B  .  C  C
   A  P  P  .  D  E  K
   .  .  .  .  D  E
   F  F  F  H  H  E
   G  G  G  I  I  .


Move 1: E-down

   A  .  B  .  .  .
   A  .  B  .  C  C
   A  P  P  .  D  .  K
   .  .  .  .  D  E
   F  F  F  H  H  E
   G  G  G  I  I  E


Move 2: P-right

   A  .  B  .  .  .
   A  .  B  .  C  C
   A  .  P  P  D  .  K
   .  .  .  .  D  E
   F  F  F  H  H  E
   G  G  G  I  I  E
```

```
Move 33: E-down

   .  .  .  .  D  .
   A  .  C  C  D  .
   A  .  B  P  P  .  K
   A  .  B  .  .  E
   F  F  F  H  H  E
   G  G  G  I  I  E


Move 34: P-right

   .  .  .  .  D  .
   A  .  C  C  D  .
   A  .  B  .  .  .  K
   A  .  B  .  .  E
   F  F  F  H  H  E
   G  G  G  I  I  E


Nodes visited: 444
Execution time: 36ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/7.txt

Loading ./test/problems/7.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 2

Initial board:

  A A A C E .
  . . B C E D
  P P B . E D K
  F . H H I I
  F J J J . G
  L L L . . G


Move 1: D-up

  A A A C E D
  . . B C E D
  P P B . E . K
  F . H H I I
  F J J J . G
  L L L . . G


Move 2: H-left

  A A A C E D
  . . B C E D
  P P B . E . K
  F H H . I I
  F J J J . G
  L L L . . G
```

```
Move 83: D-up

  F A A A . D
  F . B . . D
  . . B P P . K
  H H I I E G
  J J J C E G
  L L L C E .


Move 84: P-right

  F A A A . D
  F . B . . D
  . . B . . . K
  H H I I E G
  J J J C E G
  L L L C E .


Nodes visited: 4681
Execution time: 375ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/8.txt

Loading ./test/problems/8.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 2

Initial board:

  . . B B B A
  D D C . . A
  E . C G G A
K E . C . P P
  F F F H . .
  . . . H I I


Move 1: P-left

  . . B B B A
  D D C . . A
  E . C G G A
K E . C P P .
  F F F H . .
  . . . H I I


Move 2: A-down

  . . B B B .
  D D C . . .
  E . C G G A
K E . C P P A
  F F F H . A
  . . . H I I
```

```
Move 21: E-down

  . . C B B B
  D D C H . .
  . . C H G G
K . P P . . A
  E . F F F A
  E . . I I A


Move 22: P-left

  . . C B B B
  D D C H . .
  . . C H G G
K . . . . . A
  E . F F F A
  E . . I I A


Nodes visited: 94
Execution time: 12ms
```

4.3. A* Search

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/9.txt

Loading ./test/problems/9.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 3

Initial board:

   A  A  A  C  .  .
   B  B  B  C  .  .
   P  P  P  .  .  H  K
   .  D  .  .  .  H
   .  D  F  G  .  H
   E  E  F  G  .  .


Move 1: H-down

   A  A  A  C  .  .
   B  B  B  C  .  .
   P  P  P  .  .  .  K
   .  D  .  .  .  H
   .  D  F  G  .  H
   E  E  F  G  .  H


Move 2: P-right

   A  A  A  C  .  .
   B  B  B  C  .  .
   .  .  .  .  .  .  K
   .  D  .  .  .  H
   .  D  F  G  .  H
   E  E  F  G  .  H


Nodes visited: 7
Execution time: 1ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/10.txt

Loading ./test/problems/10.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 3

Initial board:

   A  B  G  G  G  .
   A  B  H  H  H  .
   I  I  C  D  .  .
K  .  .  C  D  P  P
   E  .  J  J  J  F
   E  .  .  L  L  F


Move 1: G-right

   A  B  .  G  G  G
   A  B  H  H  H  .
   I  I  C  D  .  .
K  .  .  C  D  P  P
   E  .  J  J  J  F
   E  .  .  L  L  F


Move 2: H-right

   A  B  .  G  G  G
   A  B  .  H  H  H
   I  I  C  D  .  .
K  .  .  C  D  P  P
   E  .  J  J  J  F
   E  .  .  L  L  F
```

```
Move 12: E-down

   A  B  C  G  G  G
   A  B  C  H  H  H
   .  I  I  D  .  .
K  .  P  P  D  .  .
   E  .  J  J  J  F
   E  L  L  .  .  F


Move 13: P-left

   A  B  C  G  G  G
   A  B  C  H  H  H
   .  I  I  D  .  .
K  .  .  .  D  .  .
   E  .  J  J  J  F
   E  L  L  .  .  F


Nodes visited: 2343
Execution time: 238ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/11.txt

Loading ./test/problems/11.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 3

Initial board:

   . . . . . .
   . . D . G F
   H H D E G F
   P P C E G . K
   A B C I I .
   A B . . . .


Move 1: C-down

   . . . . . .
   . . D . G F
   H H D E G F
   P P . E G . K
   A B C I I .
   A B C . . .


Move 2: G-up

   . . . . G .
   . . D . G F
   H H D E G F
   P P . E . . K
   A B C I I .
   A B C . . .
```

```
Move 3: E-up

   . . . E G .
   . . D E G F
   H H D . G F
   P P . . . . K
   A B C I I .
   A B C . . .


Move 4: P-right

   . . . E G .
   . . D E G F
   H H D . G F
   . . . . . . K
   A B C I I .
   A B C . . .


Nodes visited: 108
Execution time: 16ms
```

```
PS C:\Users\buege\Documents\uni\code\Stima\Tucil3_13523037> .\bin\rush_hour.exe ./test/problems/12.txt

Loading ./test/problems/12.txt ...

Choose algorithm:
1. UCS (Uniform Cost Search)
2. GBFS (Greedy Best-First Search)
3. A* Search
Choice: 3

Initial board:

       K
   A A A B . .
   . . . B C C
   D . E E G H
   D F F F G H
   D . P J J I
   . . P . . I


Move 1: E-left

       K
   A A A B . .
   . . . B C C
   D E E . G H
   D F F F G H
   D . P J J I
   . . P . . I


Move 2: B-down

       K
   A A A . . .
   . . . B C C
   D E E B G H
   D F F F G H
   D . P J J I
   . . P . . I
```

```
Move 14: E-left

       K
   .  .  .  A  A  A
   C  C  .  B  G  H
   E  E  .  B  G  H
   D  .  .  F  F  F
   D  .  P  J  J  I
   D  .  P  .  .  I


Move 15: P-up

       K
   .  .  .  A  A  A
   C  C  .  B  G  H
   E  E  .  B  G  H
   D  .  .  F  F  F
   D  .  .  J  J  I
   D  .  .  .  .  I


Nodes visited: 1886
Execution time: 185ms
```

Bab 5
Analisis Algoritma

- Universal Cost Search, Greedy Best-First Search, dan A* Search memiliki skema pencarian yang sama dengan BFS. Dengan b sebagai branching factor dan d sebagai kedalaman graf, kompleksitas algoritmanya adalah:
    - Kompleksitas waktu: O(b^d)
    - Kompleksitas ruang: O(b^d)

Bab 6
Link Repository

buege-putra/Tucil3_13523037

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	✓	