

**Laporan Tugas Kecil 3**  
**IF2211 Strategi Algoritma**  
**Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding**  
**Semester II Tahun 2024/2025**



Disusun oleh :  
Clarissa Nethania Tambunan (13523016)  
Indah Novita Tangdililing (13523047)

Dosen Pengampu:  
Dr. Nur Ulfa Maulidevi, S.T, M.Sc.  
Dr. Ir. Rinaldi Munir, M.T.  
Menterico Adrian, S.T, M.T.

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro Dan Informatika**  
**Institut Teknologi Bandung**  
**2025**

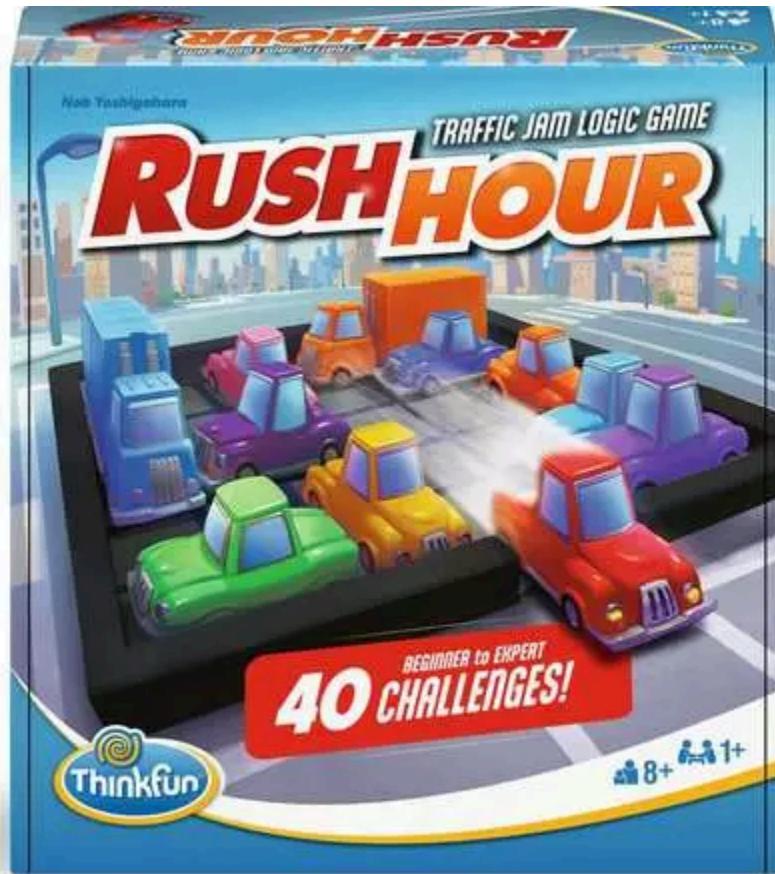
## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I</b>	
<b>DESKRIPSI TUGAS.....</b>	<b>4</b>
<b>BAB II</b>	
<b>LANDASAN TEORI.....</b>	<b>8</b>
2.1. Algoritma UCS, Greedy Best First Search, dan A*.....	8
2.2. Alur Kerja Program.....	8
<b>BAB III</b>	
<b>ANALISIS ALGORITMA DAN IMPLEMENTASI.....</b>	<b>11</b>
3.1. Algoritma Rush Hour Puzzle Solver.....	11
3.1.1. Algoritma Uniform Cost Search (UCS).....	11
3.1.2. Algoritma Greedy Best First Search.....	11
3.1.3. Algoritma A*.....	12
3.2. Notasi Pseudocode.....	12
3.2.1. AStar.java.....	12
3.2.2. GreedyBestFirst.java.....	14
3.2.3. Heuristic.java.....	16
3.2.4. HeuristicFactory.java.....	17
3.2.5. PathFinder.java.....	17
3.2.6. UCS.java.....	18
3.2.7. File Board.java.....	19
3.2.8. File Move.java.....	23
3.2.9. File Piece.java.....	23
3.2.10. File State.java.....	24
3.2.11. File FileHandler.java.....	26
3.2.12. File BoardPanel.java (GUI).....	29
3.2.13. File ControlPanel.java (GUI).....	31
3.2.14. File StatusPanel.java (GUI).....	33
3.2.15. File MainFrame.java (GUI).....	34
3.3. Analisis Algoritma UCS, Greedy Best First Search, dan A*.....	37
<b>BAB IV</b>	
<b>PENGUJIAN.....</b>	<b>38</b>
4.1. Test Case 1 - UCS.....	38
4.2. Test Case 1 - GBFS Heuristik Distance to Exit.....	40
4.3. Test Case 1 - A* Heuristik Blocked Vehicles.....	42
4.4. Test Case 1 - A* Heuristik Combined.....	44
4.5. Test Case 2 - UCS.....	46
4.6. Test Case 2 - GBFS Heuristik Blocked Vehicles.....	48
4.7. Test Case 2 - A* Heuristik Distance to Exit.....	50
4.8. Test Case 2 - A* Heuristik Combined.....	52

4.9. Test Case 3 - UCS.....	54
4.10. Test Case 3 - GBFS Heuristik Blocked Vehicles.....	56
4.11. Test Case 3 - A* Heuristik Distance to Exit.....	58
4.12. Test Case 3 - A* Heuristik Combined.....	60
4.13. Test Case 4 - UCS.....	62
4.14. Test Case 4 - GBFS Heuristik Distance to Exit.....	64
4.15. Test Case 4 - A* Heuristik Blocked Vehicles.....	66
4.16. Test Case 4 - A* Heuristik Combined.....	68
4.17. Analisis Percobaan Algoritma Pathfinding.....	70
<b>BAB V</b>	
<b>KESIMPULAN DAN SARAN.....</b>	<b>71</b>
5.1. Kesimpulan.....	71
5.2. Saran.....	72
<b>DAFTAR PUSTAKA.....</b>	<b>73</b>
<b>LAMPIRAN.....</b>	<b>73</b>

## BAB I

### DESKRIPSI TUGAS



(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

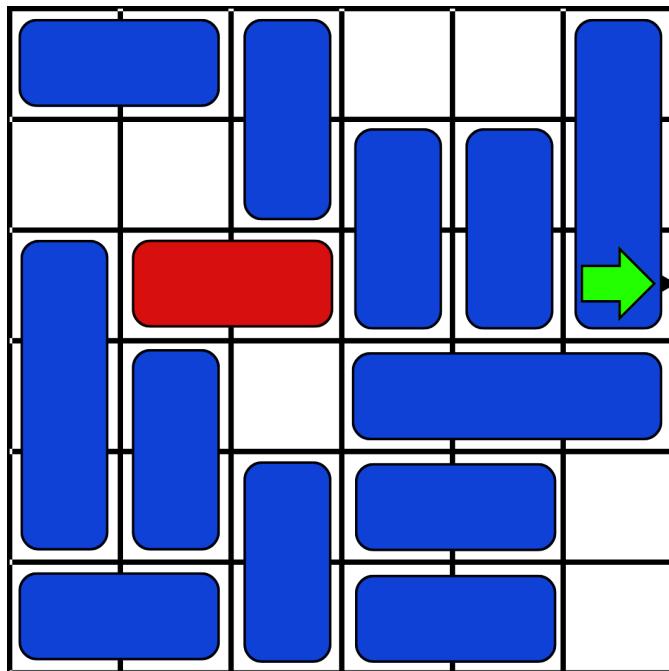
di dalam papan dengan konfigurasi tertentu berupa lokasi piece dan *orientasi*, antara *horizontal* atau *vertikal*.

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.

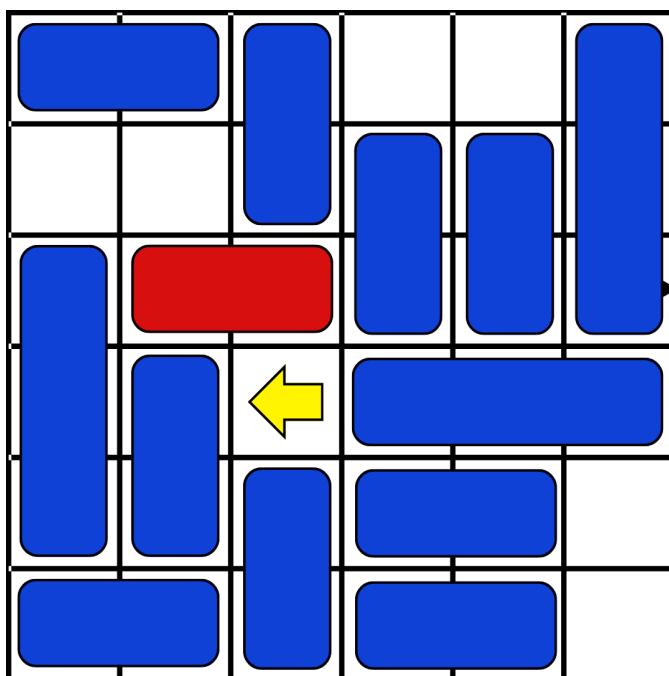
#### **Ilustrasi kasus :**

Diberikan sebuah *papan* berukuran  $6 \times 6$  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

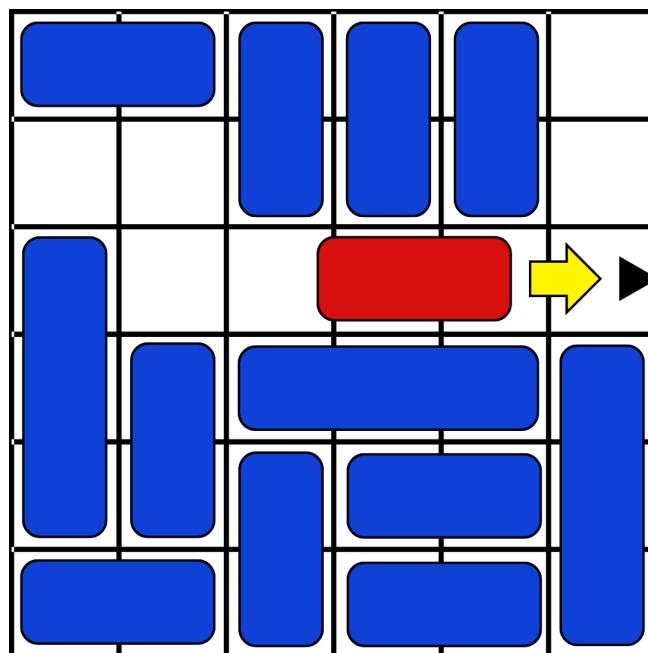
Pemain dapat menggeser-geser *piece* (termasuk *primary piece*) untuk membentuk jalan lurus antara *primary piece* dan *pintu keluar*.



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*.



Agar lebih jelas, silahkan amati video cara bermain berikut:

[The New Rush Hour by ThinkFun!](#)

Anda juga dapat melihat gif berikut untuk melihat contoh permainan [Rush Hour Solution](#).

## BAB II

### LANDASAN TEORI

#### 2.1. Algoritma UCS, Greedy Best First Search, dan A\*

Uniform Cost Search (UCS) adalah algoritma pencarian yang berfokus menemukan jalur dengan total biaya paling rendah dari titik awal sampai tujuan. UCS menggunakan struktur data antrian prioritas yang selalu memilih simpul dengan biaya terkecil yang sudah tercapai untuk diperluas lebih dulu. Algoritma ini dijamin menemukan jalur optimal dan akan selesai jika biaya setiap langkah selalu positif.

Greedy Best First Search adalah algoritma yang memilih simpul berikutnya berdasarkan perkiraan jarak terpendek ke tujuan (heuristik), tanpa memperhatikan biaya perjalanan sejauh ini. Pendekatan ini membuatnya cepat dalam mencari solusi, tetapi tidak selalu menghasilkan jalur terbaik karena hanya mengutamakan pilihan yang terlihat paling menjanjikan saat itu saja.

A\* adalah algoritma pencarian yang mengombinasikan pendekatan UCS dan Greedy Best First Search dengan menggunakan fungsi nilai  $f(n)$  yang merupakan jumlah dari biaya perjalanan sejauh ini ( $g(n)$ ) dan perkiraan biaya ke tujuan ( $h(n)$ ). Dengan cara ini, A\* mampu menemukan jalur terpendek secara efisien dan optimal, asalkan perkiraan heuristik yang digunakan tidak melebih-lebihkan biaya sebenarnya.

#### 2.2. Alur Kerja Program

1. [INPUT] konfigurasi permainan/test case dalam format ekstensi .txt. File *test case* tersebut berisi:
  - a. **Dimensi Papan** terdiri atas dua buah variabel **A** dan **B** yang membentuk papan berdimensi AxB
  - b. **Banyak piece BUKAN** *primary piece* direpresentasikan oleh variabel integer **N**.
  - c. **Konfigurasi papan** yang mencakup penempatan *piece* dan *primary piece*, serta lokasi *pintu keluar*. *Primary Piece* dilambangkan dengan huruf **P** dan pintu keluar dilambangkan dengan huruf **K**. *Piece* dilambangkan dengan huruf dan karakter selain *P* dan *K*, dan huruf/karakter berbeda melambangkan *piece* yang berbeda. *Cell kosong* dilambangkan dengan karakter '.' (titik). (**Catatan:** ingat bahwa *pintu keluar* pasti berada di *dinding* papan dan sejajar dengan orientasi *primary piece*)

File .txt yang akan dibaca memiliki format sebagai berikut:

```
A B  
N  
konfigurasi_papan
```

#### Contoh Input

6	6
11	
AAB..F	
..BCDF	
PPP CDFK	
GH.III	
GHJ...	
LLJMM.	

keterangan: "K" adalah pintu keluar, "P" adalah primary piece, Titik (".") adalah cell kosong.

Contoh konfigurasi papan lain yang mungkin berdasarkan letak *pintu keluar* (X adalah *piece/cell random*)

K XXX XXX XXX	XXX XXXX XXX	XXX XXX XXX K
------------------------	--------------------	------------------------

2. [INPUT] algoritma **pathfinding** yang digunakan
3. [INPUT] **heuristic** yang digunakan (**bonus**)
4. [OUTPUT] Banyaknya **gerakan** yang diperiksa (alias banyak 'node' yang dikunjungi)
5. [OUTPUT] Waktu eksekusi program
6. [OUTPUT] **konfigurasi papan** pada setiap tahap pergerakan/pergeseran. Output ini tidak harus diimplementasi apabila mengerjakan *bonus output GUI*. **Gunakan print berwarna** untuk menunjukkan pergerakan *piece* dengan jelas. Cukup mewarnakan **primary piece**, **pintu keluar**, dan **piece yang digerakkan** saja (boleh dengan *highlight* atau *text color*). Pastikan ketiga komponen tersebut memiliki warna berbeda.

Format sekuens adalah sebagai berikut:

Papan Awal [konfigurasi_papan_awal]
--

| Gerakan 1: [piece]-[arah gerak] [konfigurasi\_papan\_gerakan\_1] |
| Gerakan 2: [piece]-[arah gerak] [konfigurasi\_papan\_gerakan\_2] |
| Gerakan [N]: [piece]-[arah gerak] |

[konfigurasi\_papan\_gerakan\_N]  
dst

### Contoh Output

Papan Awal  
AAB..F  
. .BCDF  
GPPCDFK  
GH.III  
GHJ...  
LLJMM..

Gerakan 1: I-kiri  
AAB..F  
. .BCDF  
GPPCDFK  
GHIII.  
GHJ...  
LLJMM..

Gerakan 2: F-bawah  
AAB...  
. .BCDF  
GPPCDFK  
GHIIIF  
GHJ...  
LLJMM..

dst

Keterangan: hanya sebagai contoh. Pastikan output jelas dan mudah dimengerti. Warna dan highlight hanya untuk menunjukkan perubahan.

7. [OUTPUT] animasi gerakan-gerakan untuk mencapai solusi (**bonus GUI**).

## BAB III

### ANALISIS ALGORITMA DAN IMPLEMENTASI

#### 3.1. Algoritma Rush Hour Puzzle Solver

Berikut struktur dari source code proyek ini.

```
src/
└── algorithm/
    ├── AStar.java
    ├── GreedyBestFirst.java
    ├── Heuristic.java
    ├── HeuristicFactory.java
    ├── Pathfinder.java
    └── UCS.java
└── gui/
    ├── BoardPanel.java
    ├── ControlPanel.java
    ├── MainFrame.java
    └── StatusPanel.java
└── model/
    ├── Board.java
    ├── Move.java
    ├── Piece.java
    └── State.java
└── utility/
    └── FileHandler.java
└── Main.java
```

##### 3.1.1. Algoritma Uniform Cost Search (UCS)

Algoritma Uniform Cost Search (UCS) yang diimplementasikan dalam file UCS.java bertujuan menemukan jalur solusi dengan biaya langkah terendah secara optimal. Proses diawali dengan inisialisasi state awal papan permainan dan pembuatan priority queue (frontier) yang diurutkan berdasarkan biaya langkah. State awal dimasukkan ke dalam frontier, dan sebuah set explored dibuat untuk melacak state yang sudah dijelajahi. Dalam loop pencarian, algoritma mengambil state dengan biaya terendah dan memeriksa apakah mobil utama (P) dapat mencapai pintu keluar. Jika ya, algoritma merekonstruksi jalur solusi menggunakan parent pointers. Jika belum, state tersebut dimasukkan ke explored, lalu dilakukan ekspansi dengan menghasilkan semua pergerakan legal dari setiap kendaraan, baik secara horizontal maupun vertikal. Untuk setiap gerakan, dibuat state baru yang akan diperiksa apakah sudah dijelajahi atau belum. Jika belum, dan biaya ke state tersebut lebih rendah dari state serupa di frontier, maka state baru itu ditambahkan ke frontier. Ketika state tujuan tercapai, jalur solusi direkonstruksi dan dikembalikan dalam bentuk daftar langkah.

##### 3.1.2. Algoritma Greedy Best First Search

Algoritma Greedy Best First Search (GBFS) dalam file GreedyBestFirst.java melakukan pencarian berdasarkan estimasi heuristik ke tujuan tanpa mempertimbangkan biaya aktual langkah. Inisialisasi dimulai dengan membuat state awal dan priority queue yang diurutkan hanya berdasarkan nilai heuristik. State awal dimasukkan ke frontier, dan dibuat set explored. Dalam loop pencarian, state dengan nilai heuristik terendah diambil, dan dicek

apakah mobil utama sudah mencapai pintu keluar. Jika ya, jalur solusi dikembalikan melalui rekonstruksi dari parent pointers. Jika belum, state tersebut ditambahkan ke explored dan dilakukan ekspansi untuk menghasilkan semua pergerakan legal. Setiap pergerakan menghasilkan state baru yang kemudian diperiksa apakah sudah pernah dijelajahi. Jika belum, state tersebut langsung dimasukkan ke frontier berdasarkan nilai heuristiknya. Berbeda dengan UCS, GBFS hanya mengandalkan nilai heuristik tanpa mempertimbangkan biaya perjalanan aktual, sehingga solusi yang dihasilkan tidak dijamin optimal.

### 3.1.3. Algoritma A\*

Algoritma A\* pada file AStar.java menggabungkan pendekatan UCS dan GBFS dengan mempertimbangkan kombinasi biaya aktual ( $g$ ) dan estimasi heuristik ( $h$ ). Proses inisialisasi meliputi pembuatan state awal, priority queue berdasarkan penjumlahan  $g(n)$  dan  $h(n)$ , serta map costSoFar untuk menyimpan biaya terbaik menuju setiap state. Set explored juga dibuat untuk mencatat state yang telah dikunjungi. Dalam loop pencarian, state dengan nilai  $f(n)$  terendah diambil dari frontier dan diperiksa apakah mobil utama telah mencapai tujuan. Jika sudah, jalur solusi dikembalikan. Jika belum, state ditambahkan ke explored dan diperluas dengan menghasilkan seluruh pergerakan legal kendaraan. Untuk setiap state baru, dilakukan pengecekan apakah sudah pernah dijelajahi atau apakah biaya barunya lebih rendah dari sebelumnya. Jika ya, maka state baru dimasukkan ke frontier dan costSoFar diperbarui. Proses rekonstruksi jalur mirip seperti UCS dan GBFS. A\* menjamin solusi optimal selama heuristik yang digunakan bersifat admissible, yaitu tidak melebih-lebihkan estimasi biaya.

## 3.2. Notasi Pseudocode

### 3.2.1. AStar.java

```

Class AStar implements Pathfinder:
    private nodesVisited : integer
    private solutionStates : List<State>
    private heuristic : Heuristic

    Constructor AStar(heuristic: Heuristic):
        nodesVisited <- 0
        solutionStates <- new ArrayList<>()
        this.heuristic <- heuristic

    Function findPath(initialBoard: Board):
        nodesVisited <- 0
        solutionStates.clear()
        initialState <- new State(initialBoard, null, null, 0)

        frontier <- new PriorityQueue ordered by (state.getCost() +
heuristic.calculate(state))
        frontier.add(initialState)

        costSoFar <- new Map<String, Integer>()
    
```

```

costSoFar.put(initialState.getBoard().toString(), 0)

explored <- new Set<String>()

while frontier is not empty:
    currentState <- frontier.poll()
    nodesVisited <- nodesVisited + 1
    boardRep <- currentState.getBoard().toString()

    if currentState.getBoard().canPrimaryPieceExit() then
        -> reconstructPath(currentState)

    explored.add(boardRep)

    possibleMoves <- generatePossibleMoves(currentState.getBoard())

    for each move in possibleMoves:
        newBoard <- applyMove(currentState.getBoard(), move)
        newBoardRep <- newBoard.toString()

        if newBoardRep not in explored then
            newCost <- currentState.getCost() + move.getSteps()

            if newBoardRep not in costSoFar or newCost <
            costSoFar.get(newBoardRep) then
                costSoFar.put(newBoardRep, newCost)
                newState <- new State(newBoard, currentState, move,
                currentState.getCost() + move.getSteps())
                frontier.add(newState)

        -> new empty list

Function generatePossibleMoves(board: Board):
    possibleMoves <- new ArrayList<>()
    pieces <- board.getPieces()

    for each piece in pieces.values():
        id <- piece.getId()

        if piece.isHorizontal() then
            maxRightSteps <- getMaxStepsInDirection(board, piece, "right")
            for steps from 1 to maxRightSteps:
                possibleMoves.add(new Move(id, "right", steps))

            maxLeftSteps <- getMaxStepsInDirection(board, piece, "left")
            for steps from 1 to maxLeftSteps:
                possibleMoves.add(new Move(id, "left", steps))
        else
            maxDownSteps <- getMaxStepsInDirection(board, piece, "down")
            for steps from 1 to maxDownSteps:
                possibleMoves.add(new Move(id, "down", steps))

            maxUpSteps <- getMaxStepsInDirection(board, piece, "up")
            for steps from 1 to maxUpSteps:
                possibleMoves.add(new Move(id, "up", steps))

    -> possibleMoves

Function getMaxStepsInDirection(board: Board, piece: Piece, direction:
string):
    steps <- 0
    tempBoard <- board.copy()

```

```

canMove <- true

while canMove do
    canMove <- tempBoard.movePiece(piece.getId(), direction, 1)
    if canMove then
        steps <- steps + 1

    -> steps

Function applyMove(board: Board, move: Move):
    newBoard <- board.copy()
        newBoard.movePiece(move.getPieceId(), move.getDirection(),
move.getSteps())
    -> newBoard

Function reconstructPath(goalState: State):
    path <- new ArrayList<>()
    statePath <- new ArrayList<>()
    current <- goalState

    while current.getParent() != null:
        path.insertAtFront(current.getMoveMade())
        statePath.insertAtFront(current)
        current <- current.getParent()

    statePath.insertAtFront(current)
    solutionStates <- statePath

    -> path

Function getNodesVisited():
    -> nodesVisited

Function getSolutionStates():
    -> solutionStates

```

### 3.2.2. GreedyBestFirst.java

```

Class GreedyBestFirst implements Pathfinder:
    private nodesVisited : integer
    private solutionStates : List<State>
    private heuristic : Heuristic

    Constructor GreedyBestFirst(heuristic: Heuristic):
        nodesVisited <- 0
        solutionStates <- new ArrayList<>()
        this.heuristic <- heuristic

    Function findPath(initialBoard: Board):
        nodesVisited <- 0
        solutionStates.clear()
        initialState <- new State(initialBoard, null, null, 0)
        frontier <- new PriorityQueue ordered by heuristic.calculate(state)
        frontier.add(initialState)
        explored <- new HashSet<String>()

        while frontier is not empty do
            currentState <- frontier.poll()
            nodesVisited <- nodesVisited + 1
            boardRep <- currentState.getBoard().toString()
            explored.add(boardRep)

```

```

        if currentState.getBoard().canPrimaryPieceExit() then
            -> reconstructPath(currentState)

        possibleMoves <- generatePossibleMoves(currentState.getBoard())

        for each move in possibleMoves:
            newBoard <- applyMove(currentState.getBoard(), move)
            newBoardRep <- newBoard.toString()

            if newBoardRep not in explored then
                newState <- new State(newBoard, currentState, move,
                currentState.getCost() + move.getSteps())
                frontier.add(newState)

        -> new empty list

Function generatePossibleMoves(board: Board):
    possibleMoves <- new ArrayList<>()
    pieces <- board.getPieces()

    for each piece in pieces.values():
        id <- piece.getId()
        if piece.isHorizontal() then
            maxRightSteps <- getMaxStepsInDirection(board, piece, "right")
            for steps from 1 to maxRightSteps:
                possibleMoves.add(new Move(id, "right", steps))
            maxLeftSteps <- getMaxStepsInDirection(board, piece, "left")
            for steps from 1 to maxLeftSteps:
                possibleMoves.add(new Move(id, "left", steps))
        else
            maxDownSteps <- getMaxStepsInDirection(board, piece, "down")
            for steps from 1 to maxDownSteps:
                possibleMoves.add(new Move(id, "down", steps))
            maxUpSteps <- getMaxStepsInDirection(board, piece, "up")
            for steps from 1 to maxUpSteps:
                possibleMoves.add(new Move(id, "up", steps))

    -> possibleMoves

Function getMaxStepsInDirection(board: Board, piece: Piece, direction: string):
    steps <- 0
    tempBoard <- board.copy()
    canMove <- true
    while canMove:
        canMove <- tempBoard.movePiece(piece.getId(), direction, 1)
        if canMove then
            steps <- steps + 1
    -> steps

Function applyMove(board: Board, move: Move):
    newBoard <- board.copy()
    newBoard.movePiece(move.getPieceId(), move.getDirection(),
    move.getSteps())
    -> newBoard

Function reconstructPath(goalState: State):
    path <- new ArrayList<>()
    statePath <- new ArrayList<>()
    current <- goalState

```

```

        while current.getParent() != null do
            path.insertAtFront(current.getMoveMade())
            statePath.insertAtFront(current)
            current <- current.getParent()

            statePath.insertAtFront(current)
            solutionStates <- statePath

        -> path

Function getNodeVisited():
    -> nodesVisited

Function getSolutionStates():
    -> solutionStates

```

### 3.2.3. Heuristic.java

```

Interface Heuristic:
    Function calculate(state: State) -> integer

Class DistanceHeuristic implements Heuristic:
    Function calculate(state: State):
        board <- state.getBoard()
        primaryPiece <- board.getPrimaryPiece()

        if primaryPiece.isHorizontal() then
            row <- primaryPiece.getRow()
            exitRow <- board.getExitRow()
            exitCol <- board.getExitCol()

            if exitRow == row then
                primaryRight <- primaryPiece.getCol() +
                primaryPiece.getLength() - 1

                if exitCol > primaryRight then
                    -> exitCol - primaryRight
                else if exitCol < primaryPiece.getCol() then
                    -> primaryPiece.getCol() - exitCol

            -> board.getWidth() + board.getHeight()

Class BlockingVehiclesHeuristic implements Heuristic:
    Function calculate(state: State):
        board <- state.getBoard()
        primaryPiece <- board.getPrimaryPiece()
        blockingCount <- 0

        exitRow <- board.getExitRow()
        exitCol <- board.getExitCol()

        if primaryPiece.isHorizontal() then
            row <- primaryPiece.getRow()
            if exitRow == row then
                if exitCol > primaryPiece.getCol() then
                    startCol <- primaryPiece.getCol() +
                    primaryPiece.getLength()
                    endCol <- exitCol
                else
                    startCol <- exitCol + 1
                    endCol <- primaryPiece.getCol()

```

```

        for col from startCol to endCol - 1:
            if board.getCell(row, col) != '.' then
                blockingCount <- blockingCount + 1
            else
                -> Integer.MAX_VALUE / 2
        else
            col <- primaryPiece.getCol()
            if exitCol == col then
                if exitRow > primaryPiece.getRow() then
                    startRow <- primaryPiece.getRow() +
primaryPiece.getLength()
                    endRow <- exitRow
                else
                    startRow <- exitRow + 1
                    endRow <- primaryPiece.getRow()

                for row from startRow to endRow - 1:
                    if board.getCell(row, col) != '.' then
                        blockingCount <- blockingCount + 1
                    else
                        -> Integer.MAX_VALUE / 2
                -> blockingCount

Class CombinedHeuristic implements Heuristic:
    distanceHeuristic <- new DistanceHeuristic()
    blockingHeuristic <- new BlockingVehiclesHeuristic()

    Function calculate(state: State) -> integer:
        distance <- distanceHeuristic.calculate(state)
        blocking <- blockingHeuristic.calculate(state)

        -> distance + (2 * blocking)

```

### 3.2.4. HeuristicFactory.java

```

Class HeuristicFactory:
    Constant DISTANCE_HEURISTIC = 1
    Constant BLOCKING_VEHICLES_HEURISTIC = 2
    Constant COMBINED_HEURISTIC = 3

    Static Function createHeuristic(type: integer) -> Heuristic:
        switch (type):
            case DISTANCE_HEURISTIC:
                -> new DistanceHeuristic()
            case BLOCKING_VEHICLES_HEURISTIC:
                -> new BlockingVehiclesHeuristic()
            case COMBINED_HEURISTIC:
                -> new CombinedHeuristic()
            default:
                -> new CombinedHeuristic()

```

### 3.2.5. PathFinder.java

```

Interface PathFinder:
    Function findPath(initialBoard: Board) -> List<Move>

    Function getNodesVisited() -> integer

```

```
Function getSolutionStates() -> List<State>
```

### 3.2.6. UCS.java

```
Class UCS implements Pathfinder:
    private nodesVisited : integer
    private solutionStates : List<State>

    Constructor UCS():
        nodesVisited <- 0
        solutionStates <- new ArrayList<>()

    Function findPath(initialBoard: Board):
        nodesVisited <- 0
        solutionStates.clear()
        initialState <- new State(initialBoard, null, null, 0)
        frontier <- new PriorityQueue sorted by State.getCost()
        frontier.add(initialState)
        explored <- new HashSet<String>()

        while frontier is not empty do
            currentState <- frontier.poll()
            nodesVisited <- nodesVisited + 1
            boardRep <- currentState.getBoard().toString()
            explored.add(boardRep)

            if currentState.getBoard().canPrimaryPieceExit() then
                -> reconstructPath(currentState)

            possibleMoves <- generatePossibleMoves(currentState.getBoard())

            for each move in possibleMoves:
                newBoard <- applyMove(currentState.getBoard(), move)
                newBoardRep <- newBoard.toString()

                if newBoardRep not in explored then
                    newState <- new State(newBoard, currentState, move,
currentState.getCost() + move.getSteps())
                    shouldAdd <- true

                    for each frontierState in frontier:
                        if frontierState.getBoard().toString() == newBoardRep
and frontierState.getCost() <= newState.getCost() then
                            shouldAdd <- false
                            break

                    if shouldAdd then
                        remove from frontier any state s where
s.getBoard().toString() == newBoardRep and s.getCost() > newState.getCost()
                        frontier.add(newState)

                -> new empty list

    Function generatePossibleMoves(board: Board):
        possibleMoves <- new ArrayList<>()
        pieces <- board.getPieces()

        for each piece in pieces.values():
            id <- piece.getId()
            if piece.isHorizontal() then
```

```

        maxRightSteps <- getMaxStepsInDirection(board, piece, "right")
        for steps from 1 to maxRightSteps:
            possibleMoves.add(new Move(id, "right", steps))
        maxLeftSteps <- getMaxStepsInDirection(board, piece, "left")
        for steps from 1 to maxLeftSteps:
            possibleMoves.add(new Move(id, "left", steps))
    else:
        maxDownSteps <- getMaxStepsInDirection(board, piece, "down")
        for steps from 1 to maxDownSteps:
            possibleMoves.add(new Move(id, "down", steps))
        maxUpSteps <- getMaxStepsInDirection(board, piece, "up")
        for steps from 1 to maxUpSteps:
            possibleMoves.add(new Move(id, "up", steps))

    -> possibleMoves

Function getMaxStepsInDirection(board: Board, piece: Piece, direction: string):
    steps <- 0
    tempBoard <- board.copy()
    canMove <- true
    while canMove:
        canMove <- tempBoard.movePiece(piece.getId(), direction, 1)
        if canMove then
            steps <- steps + 1
    -> steps

Function applyMove(board: Board, move: Move):
    newBoard <- board.copy()
    newBoard.movePiece(move.getPieceId(), move.getDirection(),
move.getSteps())
    -> newBoard

Function reconstructPath(goalState: State):
    path <- new ArrayList<>()
    statePath <- new ArrayList<>()
    current <- goalState

    while current.getParent() != null do
        path.insertAtFront(current.getMoveMade())
        statePath.insertAtFront(current)
        current <- current.getParent()

    statePath.insertAtFront(current)
    solutionStates <- statePath

    -> path

Function getNodesVisited():
    -> nodesVisited

Function getSolutionStates():
    -> solutionStates

```

### 3.2.7. File Board.java

```

Class Board:
    private width : integer
    private height : integer
    private pieces : Map<Character, Piece>
    private exitRow : integer

```

```

private exitCol : integer
private exitOnRight : boolean
private exitOnBottom : boolean
private primaryPieceId : char

Constructor Board(width: integer, height: integer):
    width <- width
    height <- height
    pieces <- new HashMap<>()

Function copy():
    newBoard <- new Board(width, height)
    newBoard.exitRow <- exitRow
    newBoard.exitCol <- exitCol
    newBoard.exitOnRight <- exitOnRight
    newBoard.exitOnBottom <- exitOnBottom
    newBoard.primaryPieceId <- primaryPieceId

    for each piece in pieces.values():
        newPiece <- piece.copy()
        newBoard.pieces.put(newPiece.getId(), newPiece)

    -> newBoard

Procedure addPiece(piece: Piece):
    pieces.put(piece.getId(), piece)
    if piece.isPrimary() then
        primaryPieceId <- piece.getId()

Procedure setExit(row: integer, col: integer):
    exitRow <- row
    exitCol <- col
    exitOnRight <- (col == width - 1)
    exitOnBottom <- (row == height - 1)

Function movePiece(pieceId: char, direction: string, steps: integer):
    piece <- pieces.get(pieceId)
    if piece is null then
        -> false

    newRow <- piece.getRow()
    newCol <- piece.getCol()

    switch direction:
        case "up":
            if piece.isHorizontal() then
                -> false
            newRow <- newRow - steps
            break
        case "down":
            if piece.isHorizontal() then
                -> false
            newRow <- newRow + steps
            break
        case "left":
            if not piece.isHorizontal() then
                -> false
            newCol <- newCol - steps
            break
        case "right":
            if not piece.isHorizontal() then
                -> false

```

```

        newCol <- newCol + steps
        break
    default:
        -> false

    if not isValidMove(piece, newRow, newCol) then
        -> false

        if piece.isPrimary() and piece.isHorizontal() and exitOnRight and
newCol + piece.getLength() > width then
            if exitRow == piece.getRow() then
                piece.setCol(newCol)
                -> true
            -> false
            else if piece.isPrimary() and not piece.isHorizontal() and
exitOnBottom and newRow + piece.getLength() > height then
                if exitCol == piece.getCol() then
                    piece.setRow(newRow)
                    -> true
                -> false

            piece.setRow(newRow)
            piece.setCol(newCol)
            -> true

Function isValidMove(piece: Piece, newRow: integer, newCol: integer):
    if newRow < 0 or newCol < 0 then
        -> false

    if piece.isHorizontal():
        if newCol + piece.getLength() > width and not (piece.isPrimary()
and exitOnRight and piece.getRow() == exitRow) then
            -> false
        else:
            if newRow + piece.getLength() > height and not (piece.isPrimary()
and exitOnBottom and piece.getCol() == exitCol) then
                -> false

    r traversal [0..height - 1]:
        c traversal [0..width - 1]:
            wouldOccupy <- false
            if piece.isHorizontal():
                wouldOccupy <- r == newRow and c >= newCol and c < newCol
+ piece.getLength()
            else:
                wouldOccupy <- c == newCol and r >= newRow and r < newRow
+ piece.getLength()

            if wouldOccupy then
                for each otherPiece in pieces.values():
                    if otherPiece.getId() != piece.getId() and
otherPiece.occupies(r, c) then
                        -> false
            -> true

Function canPrimaryPieceExit():
    primaryPiece <- pieces.get(primaryPieceId)

    if primaryPiece.isHorizontal():
        if primaryPiece.getRow() != exitRow then
            -> false

```

```

        exitDirection <- 1 if exitOnRight else -1
                startCol  <- primaryPiece.getCol() + (exitDirection > 0 ?
primaryPiece.getLength() : -1)
                endCol <- exitDirection > 0 ? width : -1

        for c from startCol to endCol with step exitDirection:
            for each piece in pieces.values():
                if piece.getId() != primaryPieceId and piece.occupies(exitRow,
c) then
                    -> false

        -> true

    else:
        if primaryPiece.getCol() != exitCol then
            -> false

        exitDirection <- 1 if exitOnBottom else -1
                startRow <- primaryPiece.getRow() + (exitDirection > 0 ?
primaryPiece.getLength() : -1)
                endRow <- exitDirection > 0 ? height : -1

        for r from startRow to endRow with step exitDirection:
            for each piece in pieces.values():
                if piece.getId() != primaryPieceId and piece.occupies(r,
exitCol) then
                    -> false

        -> true

Function getCell(row: integer, col: integer) -> char:
    for each piece in pieces.values():
        if piece.occupies(row, col) then
            -> piece.getId()
    -> '.'

Function toString() -> String:
    sb <- new StringBuilder()
    r traversal [0..height - 1]:
        c traversal [0..width - 1]:
            sb.append(getCell(r, c))
            if r < height - 1 then
                sb.append('\n')
    -> sb.toString()

Function getWidth():
    -> width

Function getHeight():
    -> height

Function getPieces():
    -> pieces

Function getExitRow():
    -> exitRow

Function getExitCol():
    -> exitCol

Function getPrimaryPiece():

```

```

-> pieces.get(primaryPieceId)

Function getPrimaryPieceId():
-> primaryPieceId

```

### 3.2.8. File Move.java

```

Class Move:
    private pieceId : char
    private direction : string
    private steps : integer

    Constructor Move(pieceId: char, direction: string, steps: integer):
        pieceId <- pieceId
        direction <- direction
        steps <- steps

    Function getPieceId():
        -> pieceId

    Function getDirection():
        -> direction

    Function getSteps():
        -> steps

    Function toString():
        if steps > 1 then
            -> pieceId + "-" + direction + " " + steps
        else
            -> pieceId + "-" + direction

```

### 3.2.9. File Piece.java

```

Class Piece:
    private id : char
    private row : integer
    private col : integer
    private length : integer
    private isHorizontal : boolean
    private isPrimary : boolean

    Constructor Piece(id: char, row: integer, col: integer, length: integer,
isHorizontal: boolean, isPrimary: boolean):
        id <- id
        row <- row
        col <- col
        length <- length
        isHorizontal <- isHorizontal
        isPrimary <- isPrimary

    Function copy():
        -> new Piece(id, row, col, length, isHorizontal, isPrimary)

    Function getId():
        -> id

    Function getRow():
        -> row

```

```

Function getCol():
    -> col

Procedure setRow(row: integer):
    this.row <- row

Procedure setCol(col: integer):
    this.col <- col

Function getLength():
    -> length

Function isHorizontal():
    -> isHorizontal

Function isPrimary():
    -> isPrimary

Function occupies(r: integer, c: integer):
    if isHorizontal then
        -> (r == row) and (c >= col) and (c < col + length)
    else
        -> (c == col) and (r >= row) and (r < row + length)

Function toString():
    str <- id
    if isPrimary then
        str <- str + "(P)"
    str <- str + "[" + row + "," + col + "]"
    -> str

```

### 3.2.10. File State.java

```

Class State:
    private board : Board
    private parent : State
    private moveMade : Move
    private cost : integer
    private heuristicValue : integer

    Constructor State(board: Board, parent: State, moveMade: Move, cost: integer):
        board <- board
        parent <- parent
        moveMade <- moveMade
        cost <- cost
        heuristicValue <- 0

    Function getBoard():
        -> board

    Function getParent():
        -> parent

    Function getMoveMade():
        -> moveMade

    Function getCost():
        -> cost

```

```

Function getHeuristicValue():
    -> heuristicValue

Procedure setHeuristicValue(value: integer):
    heuristicValue <- value

Function getF():
    -> cost + heuristicValue

Function generateNextStates():
    nextStates <- new List<State>()

    for each piece in board.getPieces().values():
        if piece.isHorizontal() then
            {move left}
            for steps from 1 to board.getWidth():
                newBoard <- board.copy()
                if newBoard.movePiece(piece.getId(), "left", steps) then
                    move <- new Move(piece.getId(), "left", steps)
                    nextStates.add(new State(newBoard, this, move, cost +
1))
                else
                    break

            {move right}
            for steps from 1 to board.getWidth():
                newBoard <- board.copy()
                if newBoard.movePiece(piece.getId(), "right", steps) then
                    move <- new Move(piece.getId(), "right", steps)
                    nextStates.add(new State(newBoard, this, move, cost +
1))
                else
                    break
            else
                {move up}
                for steps from 1 to board.getHeight():
                    newBoard <- board.copy()
                    if newBoard.movePiece(piece.getId(), "up", steps) then
                        move <- new Move(piece.getId(), "up", steps)
                        nextStates.add(new State(newBoard, this, move, cost +
1))
                    else
                        break

                {move down}
                for steps from 1 to board.getHeight():
                    newBoard <- board.copy()
                    if newBoard.movePiece(piece.getId(), "down", steps) then
                        move <- new Move(piece.getId(), "down", steps)
                        nextStates.add(new State(newBoard, this, move, cost +
1))
                    else
                        break
        else
            break
    else
        break

    -> nextStates

Function isSolution():
    -> board.canPrimaryPieceExit()

Function getStateKey():
    -> board.toString()

```

```

Function equals(obj: Object):
    if this == obj then
        -> true
    if obj == null or getClass() != obj.getClass() then
        -> false
    other <- (State) obj
    -> getStateKey() == other.getStateKey()

Function hashCode():
    -> getStateKey().hashCode()

```

### 3.2.11. File FileHandler.java

```

Class FileHandler:
    Function loadBoardFromFile(filePath: string) throws IOException:
        reader <- new BufferedReader(new FileReader(filePath))

        try:
            line <- reader.readLine()
            if line == null then
                throw IOException("File is empty or incomplete")

            dimensions <- line.split(" ")
            if length(dimensions) < 2 then
                throw IOException("Invalid board dimensions format. Expected
'width height'")

            width <- Integer.parseInt(dimensions[0])
            height <- Integer.parseInt(dimensions[1])

            line <- reader.readLine() {skip number of pieces line}

            board <- new Board(width, height)

            grid <- new char[height][width]
            r traversal [0..height - 1]:
                line <- reader.readLine()
                if line == null or length(line) < width then
                    throw IOException("Invalid grid row " + (r + 1) + ":" +
insufficient length")
                c traversal [0..width - 1]:
                    grid[r][c] <- line.charAt(c)

            exitRow <- -1
            exitCol <- -1
            primaryRow <- -1

            {Find primary piece 'P' row}
            r traversal [0..height - 1]:
                c traversal [0..width - 1]:
                    if grid[r][c] == 'P' then
                        primaryRow <- r
                        break
                    if primaryRow >= 0 then break

            {Find exit 'K' in same row as 'P'}
            if primaryRow >= 0 then
                for c from 0 to width - 1:
                    if grid[primaryRow][c] == 'K' then
                        exitRow <- primaryRow
                        exitCol <- c

```

```

        grid[exitRow][exitCol] <- '.'
        break

{If 'K' not found, find 'K' anywhere}
if exitRow < 0 then
    r traversal [0..height - 1]:
        c traversal [0..width - 1]:
            if grid[r][c] == 'K' then
                exitRow <- r
                exitCol <- c
                grid[r][c] <- '.'
                break
            if exitRow >= 0 then break

{Default exit if still not found}
if exitRow < 0 then
    exitRow <- if primaryRow >= 0 then primaryRow else height / 2
    exitCol <- width - 1

board.setExit(exitRow, exitCol)

pieceBounds <- new Map<Character, int[]>()

{Find pieces and their boundaries}
r traversal [0..height - 1]:
    c traversal [0..width - 1]:
        id <- grid[r][c]
        if id != '.' then
            if not pieceBounds.containsKey(id) then
                pieceBounds.put(id, [r, c, r, c])
            else
                bounds <- pieceBounds.get(id)
                bounds[2] <- max(bounds[2], r)
                bounds[3] <- max(bounds[3], c)

{Create pieces and add to board}
for each entry in pieceBounds:
    id <- entry.key
    bounds <- entry.value
    startRow <- bounds[0]
    startCol <- bounds[1]
    endRow <- bounds[2]
    endCol <- bounds[3]

    isHorizontal <- (startRow == endRow)
    length <- if isHorizontal then (endCol - startCol + 1) else
    (endRow - startRow + 1)
    isPrimary <- (id == 'P')

    piece <- new Piece(id, startRow, startCol, length,
isHorizontal, isPrimary)
    board.addPiece(piece)

-> board

finally:
    reader.close()

Procedure printBoard(board: Board, movingPieceId: char):
    height <- board.getHeight()
    width <- board.getWidth()
    exitRow <- board.getExitRow()

```

```

exitCol <- board.getExitCol()

RESET <- "\u001B[0m"
RED <- "\u001B[91m"
GREEN <- "\u001B[92m"

{Print top border}
print "+"
c traversal [0..width - 1]:
    print "---+"
println()

r traversal [0..height - 1]:
    print "|"
    c traversal [0..width - 1]:
        cell <- board.getCell(r, c)
        color <- ""
        if cell == movingPieceId then
            color <- GREEN
        else if cell == 'P' then
            color <- RED

        print " " + color + cell + RESET + " "

        if c == width - 1 and r == exitRow then
            print " "
        else
            print "|"
    println()

{Print row separator}
print "+"
c traversal [0..width - 1]:
    print "---"
println()

Procedure saveSolutionToFile(solution: List<Move>, boardStates: List<Board>, filePath: string, timeTaken: long, nodesVisited: integer) throws IOException:
    if not filePath.endsWith(".txt") then
        throw IOException("Invalid file format. The file must be a .txt file.")

    writer <- new BufferedWriter(new FileWriter(filePath))
    try:
        writer.write("=====\\n")
        writer.write("===== SOLUTION FOUND\\n")
        writer.write("=====\\n")

        if solution.isEmpty() then
            writer.write("\\nNo solution found.")
            return
        else
            writer.write("\\nSolution path contains " + solution.size() + " moves.\\n")

        writer.write("Step by step solution:\\n")
        i traversal [0..solution.size() - 1]:
            move <- solution.get(i)
            writer.write("\\nStep " + (i + 1) + ": " + move + "\\n")
            currentBoard <- boardStates.get(i)
            height <- currentBoard.getHeight()

```

```

width <- currentBoard.getWidth()
exitRow <- currentBoard.getExitRow()
exitCol <- currentBoard.getExitCol()

{Print top border}
writer.write("+")
c traversal [0 to width - 1]:
    writer.write("---+")
writer.write("\n")

r traversal [0..height - 1]:
    writer.write("|")
    c traversal [0..width - 1]:
        cell <- currentBoard.getCell(r, c)
        writer.write(" " + cell + " ")
        if c == width - 1 and r == exitRow then
            writer.write(" ")
        else
            writer.write("|")
    writer.write("\n")

{Print row separator}
writer.write("+")
c traversal [0..width - 1]:
    writer.write("---+")
writer.write("\n")

writer.write("\n=====\n")
writer.write("                      STATISTICS\n")
writer.write("=====\n")
writer.write("Path length: " + solution.size() + " moves\n")
writer.write("Nodes visited: " + nodesVisited + "\n")
writer.write("Execution time: " + timeTaken + " ms\n")
writer.write("=====\n")

finally:
    writer.close()

```

### 3.2.12. File BoardPanel.java (GUI)

```

Class BoardPanel extends JPanel:
    private board : Board
    private pieceColors : Map<Character, Color>
    private static CELL_SIZE : integer = 50

    Constructor BoardPanel():
        setPreferredSize(new Dimension(600, 600))

    Function setBoard(board: Board):
        this.board <- board
        assignColors()
        width <- board.getWidth()
        height <- board.getHeight()
        setPreferredSize(new Dimension(width * CELL_SIZE, height * CELL_SIZE))
        revalidate()
        repaint()

    Function assignColors():
        pieceColors.clear()
        if board is null then
            return

```

```

        colors <- [Color.RED, Color.BLUE, Color.GREEN.darker(), Color.ORANGE,
Color.MAGENTA, Color.CYAN.darker(), Color.PINK, Color.YELLOW.darker()]
        idx <- 0
        for each id in board.getPieces().keySet():
            pieceColors.put(id, colors[idx % colors.length])
            idx <- idx + 1

Override Function paintComponent(g: Graphics):
    super.paintComponent(g)
    if board is null then
        return

    width <- board.getWidth()
    height <- board.getHeight()
    g2 <- (Graphics2D) g
    oldStroke <- g2.getStroke()

    r traversal [0..height - 1]:
        c traversal [0..width - 1]:
            cellChar <- board.getCell(r, c)
            bgColor <- Color.LIGHT_GRAY
            if cellChar == '.' then
                bgColor <- Color.WHITE
            else if cellChar == 'P' then
                bgColor <- Color.RED
            else if pieceColors contains key cellChar then
                bgColor <- pieceColors.get(cellChar)

            g.setColor(bgColor)
            g.fillRect(c * CELL_SIZE, r * CELL_SIZE, CELL_SIZE, CELL_SIZE)
            g.setColor(Color.BLACK)
            g.drawRect(c * CELL_SIZE, r * CELL_SIZE, CELL_SIZE, CELL_SIZE)

            if cellChar != '.':
                g.setColor(Color.BLACK)
                g.setFont(g.getFont().deriveFont(Font.BOLD, 20f))
                fm <- g.getFontMetrics()
                text <- String.valueOf(cellChar)
                textWidth <- fm.stringWidth(text)
                textHeight <- fm.getAscent()
                x <- c * CELL_SIZE + (CELL_SIZE - textWidth) / 2
                y <- r * CELL_SIZE + (CELL_SIZE + textHeight) / 2 - 4
                g.drawString(text, x, y)

    exitRow <- board.getExitRow()
    exitCol <- board.getExitCol()
    g2.setStroke(new BasicStroke(5))
    g2.setColor(Color.GREEN.darker())

    if exitRow >= 0 and exitRow < height and exitCol == -1 then
        x <- 0
        y <- exitRow * CELL_SIZE
        g2.drawLine(x, y, x, y + CELL_SIZE)
    else if exitRow >= 0 and exitRow < height and exitCol == width then
        x <- width * CELL_SIZE - 1
        y <- exitRow * CELL_SIZE
        g2.drawLine(x, y, x, y + CELL_SIZE)
    else if exitCol >= 0 and exitCol < width and exitRow == -1 then
        x <- exitCol * CELL_SIZE
        y <- 0
        g2.drawLine(x, y, x + CELL_SIZE, y)
    else if exitCol >= 0 and exitCol < width and exitRow == height then

```

```

        x <- exitCol * CELL_SIZE
        y <- height * CELL_SIZE - 1
        g2.drawLine(x, y, x + CELL_SIZE, y)

    g2.setStroke(oldStroke)

```

### 3.2.13. File ControlPanel.java (GUI)

```

Class ControlPanel extends JPanel:
    private filePathField : JTextField
    private browseButton : JButton
    private algorithmComboBox : JComboBox<String>
    private heuristicComboBox : JComboBox<String>
    private solveButton : JButton
    private playButton : JButton
    private pauseButton : JButton
    private prevButton : JButton
    private nextButton : JButton
    private saveButton : JButton // tambahan tombol Save opsional

    private browseAction : Consumer<Void>
    private solveAction : Consumer<Void>
    private playAction : Consumer<Void>
    private pauseAction : Consumer<Void>
    private prevAction : Consumer<Void>
    private nextAction : Consumer<Void>

    Constructor ControlPanel(browseCallback: Runnable, solveCallback: Runnable,
    Runnable,
                    playCallback: Runnable, pauseCallback: Runnable,
                    prevCallback: Runnable, nextCallback: Runnable,
                    saveButton: JButton):
        this.saveButton <- saveButton
        this.browseAction <- lambda v: browseCallback.run()
        this.solveAction <- lambda v: solveCallback.run()
        this.playAction <- lambda v: playCallback.run()
        this.pauseAction <- lambda v: pauseCallback.run()
        this.prevAction <- lambda v: prevCallback.run()
        this.nextAction <- lambda v: nextCallback.run()
        initComponents()
        layoutComponents()

Procedure initComponents():
    filePathField <- new JTextField(30)
    filePathField.setEditable(false)

    browseButton <- new JButton("Browse")
    browseButton.addActionListener(e -> browseAction.accept(null))

    algorithmComboBox <- new JComboBox(["Uniform Cost Search (UCS)",
    "Greedy Best First Search (GBFS)", "A* Search"])
    algorithmComboBox.addActionListener(e -> {
        idx <- algorithmComboBox.getSelectedIndex()
        heuristicComboBox.setEnabled(idx == 1 or idx == 2)
    })

    heuristicComboBox <- new JComboBox(["Distance to Exit", "Blocking
    Vehicles", "Combined (Distance + Blocking Vehicles)"])
    heuristicComboBox.setEnabled(false)

    solveButton <- new JButton("Solve")

```

```

solveButton.addActionListener(e -> solveAction.accept(null))

playButton <- new JButton("Play")
playButton.setEnabled(false)
playButton.addActionListener(e -> playAction.accept(null))

pauseButton <- new JButton("Pause")
pauseButton.setEnabled(false)
pauseButton.addActionListener(e -> pauseAction.accept(null))

prevButton <- new JButton("Prev")
prevButton.setEnabled(false)
prevButton.addActionListener(e -> prevAction.accept(null))

nextButton <- new JButton("Next")
nextButton.setEnabled(false)
nextButton.addActionListener(e -> nextAction.accept(null))

Procedure layoutComponents():
    setLayout(FlowLayout(LEFT))
    add(new JLabel("Puzzle file:"))
    add(filePathField)
    add(browseButton)
    add(new JLabel("Algorithm:"))
    add(algorithmComboBox)
    add(new JLabel("Heuristic:"))
    add(heuristicComboBox)
    add(solveButton)
    add(playButton)
    add(pauseButton)
    add(prevButton)
    add(nextButton)
    if saveButton != null then
        add(saveButton)

Procedure addSaveButton(button: JButton):
    saveButton <- button
    add(saveButton)
    revalidate()
    repaint()

Procedure setFilePath(path: string):
    filePathField.setText(path)

Function getSelectedAlgorithmIndex():
    -> algorithmComboBox.getSelectedIndex()

Function getSelectedHeuristicIndex():
    -> heuristicComboBox.getSelectedIndex()

Procedure setControlsEnabled(enabled: boolean):
    solveButton.setEnabled(enabled)
    playButton.setEnabled(enabled)
    pauseButton.setEnabled(false)
    prevButton.setEnabled(enabled)
    nextButton.setEnabled(enabled)
    if saveButton != null then
        saveButton.setEnabled(enabled)

Procedure setPlaying(isPlaying: boolean):
    playButton.setEnabled(not isPlaying)
    pauseButton.setEnabled(isPlaying)

```

```

solveButton.setEnabled(not isPlaying)
browseButton.setEnabled(not isPlaying)
prevButton.setEnabled(not isPlaying)
nextButton.setEnabled(not isPlaying)
if saveButton != null then
    saveButton.setEnabled(not isPlaying)

```

### 3.2.14. File StatusPanel.java (GUI)

```

Class StatusPanel extends JPanel:
    private statusLabel : JLabel
    private prevButton : JButton
    private nextButton : JButton
    private saveButton : JButton

    Constructor StatusPanel(prevCallback: Runnable, nextCallback: Runnable,
    saveCallback: Runnable):
        setLayout(BorderLayout(horizontalGap=10, verticalGap=0))
        setBorder(emptyBorder(top=5, left=10, bottom=5, right=10))

        statusLabel <- new JLabel("Load a puzzle to start.")
        statusLabel.setFont("SansSerif", PLAIN, 16)
        statusLabel.setHorizontalTextPosition(LEFT)
        statusLabel.setPreferredSize(width=500, height=25)

        buttonPanel <- new JPanel(new FlowLayout(RIGHT, hgap=10, vgap=0))

        prevButton <- new JButton("Prev")
        prevButton.setEnabled(false)
        prevButton.addActionListener(e -> prevCallback.run())

        nextButton <- new JButton("Next")
        nextButton.setEnabled(false)
        nextButton.addActionListener(e -> nextCallback.run())

        saveButton <- new JButton("Save")
        saveButton.setEnabled(false)
        saveButton.addActionListener(e -> saveCallback.run())

        buttonPanel.add(prevButton)
        buttonPanel.add(nextButton)
        buttonPanel.add(saveButton)

        add(statusLabel, BorderLayout.CENTER)
        add(buttonPanel, BorderLayout.EAST)

Procedure setStatus(status: string):
    statusLabel.setText(status)

Procedure setPrevEnabled(enabled: boolean):
    prevButton.setEnabled(enabled)

Procedure setNextEnabled(enabled: boolean):
    nextButton.setEnabled(enabled)

Procedure setSaveEnabled(enabled: boolean):
    saveButton.setEnabled(enabled)

Function getPreferredSize():
    baseSize <- super.getPreferredSize()
    -> new Dimension(baseSize.width, 40)

```

### 3.2.15. File MainFrame.java (GUI)

```
Class MainFrame extends JFrame:  
    private controlPanel : ControlPanel  
    private boardPanel : BoardPanel  
    private statusPanel : StatusPanel  
  
    private currentBoard : Board  
    private solutionMoves : List<Move>  
    private boardStates : List<Board>  
    private pathfinder : Pathfinder  
  
    private animationTimer : Timer  
    private animationStep : integer = 0  
  
    private saveButton : JButton  
  
    private startTime : long  
    private endTime : long  
  
    Constructor MainFrame():  
        setTitle("Rush Hour Puzzle Solver")  
        setDefaultCloseOperation(EXIT_ON_CLOSE)  
        setLayout(BorderLayout)  
        setSize(900, 700)  
        setLocationRelativeTo(null)  
  
        saveButton <- new JButton("Save Solution")  
        saveButton.setEnabled(false)  
        saveButton.addActionListener(e -> onSaveClicked())  
  
        controlPanel <- new ControlPanel(onBrowseClicked, onSolveClicked,  
onPlayClicked, onPauseClicked, onPrevClicked, onNextClicked, saveButton)  
        boardPanel <- new BoardPanel()  
        statusPanel <- new StatusPanel(onPrevClicked, onNextClicked,  
onSaveClicked)  
  
        add(controlPanel, BorderLayout.NORTH)  
        add(new JScrollPane(boardPanel), BorderLayout.CENTER)  
        add(statusPanel, BorderLayout.SOUTH)  
  
        animationTimer <- new Timer(800, e -> {  
            if boardStates != null and animationStep < size(boardStates) - 1  
        then  
            animationStep <- animationStep + 1  
            updateBoardState(animationStep)  
        else  
            animationTimer.stop()  
            controlPanel.setPlaying(false)  
        })  
  
    Procedure onSaveClicked():  
        if solutionMoves == null or solutionMoves is empty then  
            showMessage("No solution to save!", "Warning")  
            return  
  
        chooser <- new JFileChooser()  
        chooser.setFileFilter("Text Files", "txt")  
        chooser.setDialogTitle("Save Solution")
```

```

        result <- chooser.showSaveDialog(this)

        if result == APPROVE_OPTION then
            filePath <- chooser.getSelectedFile().getAbsolutePath()
            if not filePath.endsWith(".txt") then
                filePath <- filePath + ".txt"

            try
                timeTaken <- endTime - startTime
                FileHandler.saveSolutionToFile(solutionMoves, boardStates,
filePath, timeTaken, pathfinder.getNodesVisited())
                showMessage("Solution saved successfully to " + filePath,
"Success")
            catch IOException as ex
                showMessage("Failed to save solution: " + ex.getMessage(),
"Error")

Procedure onBrowseClicked():
    chooser <- new JFileChooser()
    chooser.setFileFilter("Text Files", "txt")
    result <- chooser.showOpenDialog(this)

    if result == APPROVE_OPTION then
        file <- chooser.getSelectedFile()
        controlPanel.setFilePath(file.getAbsolutePath())
        try
            currentBoard <-
FileHandler.loadBoardFromFile(file.getAbsolutePath())
            boardPanel.setBoard(currentBoard)
            statusPanel.setStatus("Puzzle loaded. Ready to solve.")
            resetAnimation()
        catch Exception as ex
            showMessage("Failed to load puzzle:\n" + ex.getMessage(),
"Error")

Procedure onSolveClicked():
    if currentBoard == null then
        showMessage("Please load a puzzle file first.", "Warning")
        return

    algoIndex <- controlPanel.getSelectedAlgorithmIndex()
    heuristicIndex <- controlPanel.getSelectedHeuristicIndex() + 1

    switch algoIndex:
        case 0:
            pathfinder <- new UCS()
        case 1:
            GreedyBestFirst(HeuristicFactory.createHeuristic(heuristicIndex))
        case 2:
            AStar(HeuristicFactory.createHeuristic(heuristicIndex))
        default:
            showMessage("Invalid algorithm selection.", "Error")
            return

    statusPanel.setStatus("Solving...")
    controlPanel.setControlsEnabled(false)
    saveButton.setEnabled(false)
    statusPanel.setPrevEnabled(false)
    statusPanel.setNextEnabled(false)
    statusPanel.setSaveEnabled(false)

```

```

        worker <- new SwingWorker<Void, Void>():
            solution : List<Move>
            states : List<State>

            override doInBackground():
                startTime <- currentTimeMillis()
                solution <- pathfinder.findPath(currentBoard)
                endTime <- currentTimeMillis()
                states <- pathfinder.getSolutionStates()
                return null

            override done():
                try
                    get() // check exceptions
                    solutionMoves <- solution
                    boardStates <- map(states, state -> state.getBoard())

                    if solutionMoves is empty then
                        statusPanel.setStatus("No solution found.")
                        showMessage("No solution found!", "Info")
                        resetAnimation()
                    else
                        statusPanel.setStatus("Solution found with " +
size(solutionMoves) + " moves, nodes visited: " +
pathfinder.getNodesVisited() + ", "
execution time: " + (endTime - startTime) + " ms")
                        animationStep <- 0
                        boardPanel.setBoard(boardStates[0])
                        controlPanel.setControlsEnabled(true)
                        controlPanel.setPlaying(false)
                        saveButton.setEnabled(true)
                        statusPanel.setPrevEnabled(false)
                        statusPanel.setNextEnabled(size(boardStates) > 1)
                        statusPanel.setSaveEnabled(true)
                catch Exception as e
                    showMessage("Error during solving:\n" + e.getMessage(),
"Error")
                    resetAnimation()

            worker.execute()

Procedure onPlayClicked():
    animationTimer.start()
    controlPanel.setPlaying(true)
    statusPanel.setPrevEnabled(false)
    statusPanel.setNextEnabled(false)

Procedure onPauseClicked():
    animationTimer.stop()
    controlPanel.setPlaying(false)
    statusPanel.setPrevEnabled(animationStep > 0)
    statusPanel.setNextEnabled(boardStates != null and animationStep <
size(boardStates) - 1)

Procedure onNextClicked():
    if boardStates == null or animationStep >= size(boardStates) - 1 then
        return
    animationStep <- animationStep + 1
    updateBoardState(animationStep)

Procedure onPrevClicked():

```

```

        if boardStates == null or animationStep <= 0 then return
        animationStep <- animationStep - 1
        updateBoardState(animationStep)

Procedure updateBoardState(step: integer):
    boardPanel.setBoard(boardStates[step])
    statusPanel.setStatus("Step " + step + " / " + (size(boardStates) -
1))
    statusPanel.setPrevEnabled(step > 0)
    statusPanel.setNextEnabled(step < size(boardStates) - 1)

Procedure resetAnimation():
    animationTimer.stop()
    animationStep <- 0
    solutionMoves <- null
    boardStates <- null
    controlPanel.setControlsEnabled(true)
    controlPanel.setPlaying(false)
    saveButton.setEnabled(false)
    statusPanel.setPrevEnabled(false)
    statusPanel.setNextEnabled(false)
    statusPanel.setSaveEnabled(false)

    if currentBoard != null then
        boardPanel.setBoard(currentBoard)
    else
        boardPanel.setBoard(null)

    statusPanel.setStatus("Load a puzzle to start.")

Procedure main(args):
    SwingUtilities.invokeLater(() -> new MainFrame().setVisible(true))

```

### 3.3. Analisis Algoritma UCS, Greedy Best First Search, dan A\*

Dalam algoritma pencarian seperti UCS, Greedy Best First Search, dan A\*, fungsi evaluasi  $f(n)$  sangat penting untuk menentukan urutan perluasan simpul. Fungsi ini biasanya didefinisikan sebagai  $f(n) = g(n) + h(n)$ , di mana  $g(n)$  adalah biaya yang sudah dikeluarkan untuk mencapai simpul  $n$ , dan  $h(n)$  adalah estimasi biaya dari  $n$  menuju tujuan. Dengan definisi ini, masing-masing algoritma memiliki cara berbeda dalam menghitung  $f(n)$ : jika  $f(n) = g(n)$ , maka algoritmanya adalah Uniform Cost Search (UCS) yang hanya memperhitungkan biaya jalur sejauh ini; jika  $f(n) = h(n)$ , maka algoritmanya Greedy Best First Search yang hanya mengandalkan estimasi heuristik tanpa mempedulikan biaya yang sudah ditempuh; dan jika  $f(n) = g(n) + h(n)$ , maka algoritmanya adalah A\* yang menggabungkan keduanya.

Heuristik yang digunakan dalam A\* disebut *admissible* jika tidak pernah melebih-lebihkan biaya sebenarnya dari simpul  $n$  ke tujuan, artinya nilai  $h(n)$  selalu lebih kecil atau sama dengan biaya sesungguhnya untuk mencapai tujuan dari  $n$ . Heuristik admissible ini penting agar A\* dapat menjamin menemukan solusi optimal, karena algoritma tidak akan melewatkannya solusi terbaik akibat perkiraan biaya yang terlalu optimis. Dengan kata lain, heuristik admissible memberikan batas bawah realistik yang menjaga keakuratan pencarian.

Dalam konteks permainan Rush Hour, jika semua langkah memiliki biaya yang sama, UCS akan berperilaku sama dengan Breadth-First Search (BFS) karena keduanya memperluas simpul berdasarkan jarak langkah yang sudah ditempuh tanpa membedakan biaya per langkah. Namun, jika biaya langkah berbeda, UCS dapat menghasilkan urutan simpul yang diperluas dan jalur yang berbeda dari BFS, karena UCS selalu memilih jalur dengan biaya terendah sejauh ini.

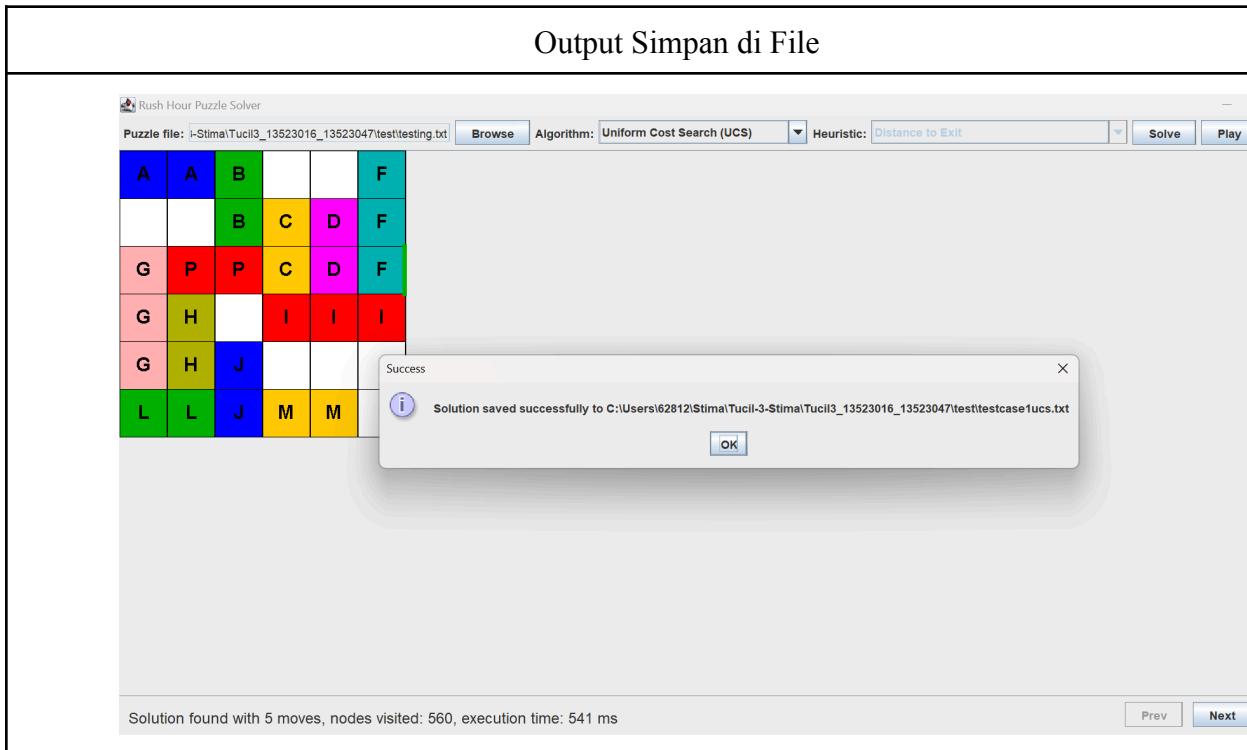
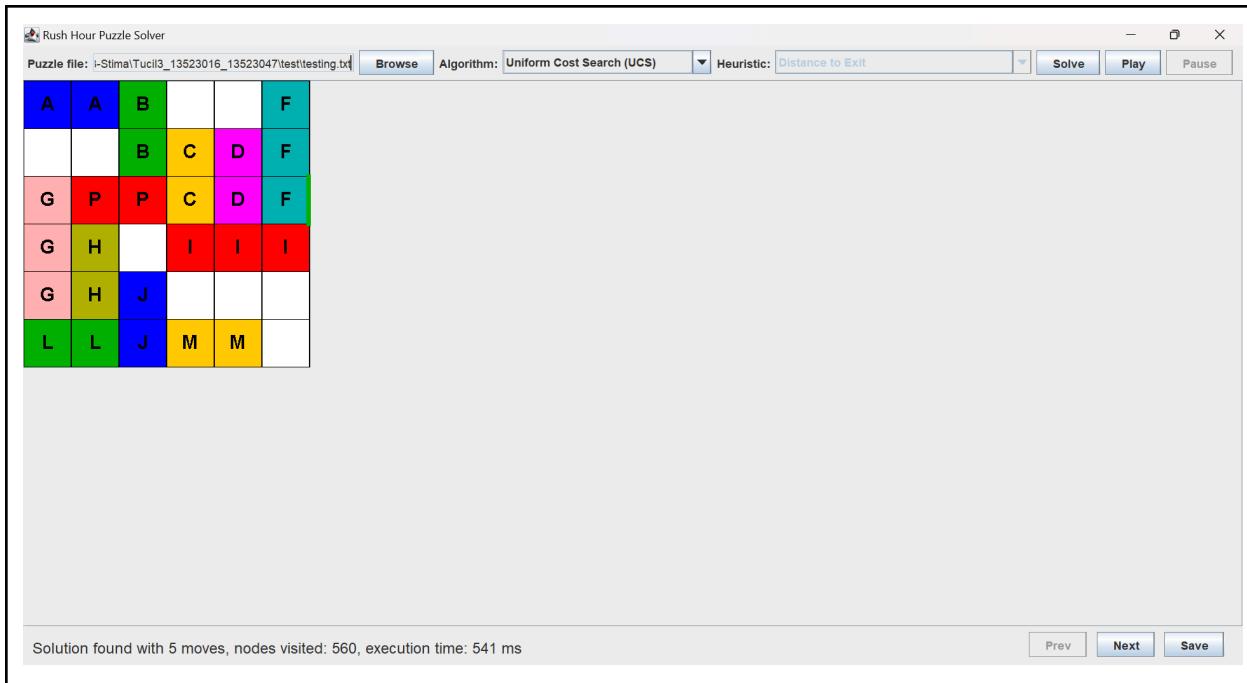
Secara teoritis, A\* lebih efisien daripada UCS dalam menyelesaikan Rush Hour, karena A\* memanfaatkan heuristik untuk mengarahkan pencarian ke tujuan sehingga mengurangi jumlah simpul yang harus diperiksa. UCS hanya mempertimbangkan biaya kumulatif tanpa petunjuk menuju tujuan, sehingga berpotensi menjelajah banyak simpul yang tidak relevan. Dengan heuristik yang baik dan *admissible*, A\* dapat memangkas ruang pencarian secara signifikan, mempercepat proses pencarian solusi.

Di sisi lain, Greedy Best First Search mengandalkan heuristik saja tanpa mempertimbangkan biaya jalur yang sudah ditempuh, sehingga meskipun sering lebih cepat dalam menemukan solusi, algoritma ini tidak menjamin solusi yang optimal. Dalam Rush Hour, hal ini berarti Greedy bisa menemukan jalan keluar dengan cepat, tetapi jalur yang ditemukan mungkin tidak paling efisien atau paling sedikit langkahnya. Oleh karena itu, Greedy Best First Search lebih cocok untuk situasi di mana kecepatan pencarian lebih diutamakan daripada optimalitas hasil.

## BAB IV PENGUJIAN

### 4.1. Test Case 1 - UCS

Input : test/testing.txt
6 6 11 AAB..F .BCDF GPPCDFK GH.III GHJ... LLJMM.
Output GUI



```

File Edit Selection View ... ← → ⌂ TuCI3_13523016_13523047
① README.md test > testcaseTuCS.txt U
1 =====
2 | *** RUSH HOUR PUZZLE SOLVER ***
3 =====
4
5 Initial Board:
6 +---+---+---+---+
7 | A | A | B | . | F |
8 +---+---+---+---+
9 | . | . | B | C | D | F |
10 +---+---+---+---+
11 | G | P | P | C | D | F |
12 +---+---+---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+---+
15 | G | H | D | . | . | .
16 +---+---+---+---+
17 | L | L | J | M | M | . |
18 +---+---+---+---+
19 =====
20 | | | | | SOLUTION FOUND
21 =====
22
23 Solution path contains 5 moves:
24
25 Step by step solution:
26
27 Step 1: C-up
28 +---+---+---+---+
29 | A | A | B | C | . | F |
30 +---+---+---+---+
31 | . | . | B | C | D | F |
32 +---+---+---+---+
33 | G | P | P | I | . | D | F |
34 +---+---+---+---+
35 | G | H | . | I | I | I |
36 +---+---+---+---+
Ln 13, Col 26 Spaces: 6 UTF-8 LF Plain Text

File Edit Selection View ... ← → ⌂ TuCI3_13523016_13523047
① README.md test > testcaseTuCS.txt U
77 +---+---+---+---+
78 | G | P | P | . | . | .
79 +---+---+---+---+
80 | G | H | I | I | I | F |
81 +---+---+---+---+
82 | G | H | J | . | . | F |
83 +---+---+---+---+
84 | L | L | J | M | M | F |
85 +---+---+---+---+
86
87 Step 5: P-right 3
88 +---+---+---+---+
89 | A | A | B | C | D | .
90 +---+---+---+---+
91 | . | . | B | C | D | .
92 +---+---+---+---+
93 | G | . | . | . | P | P |
94 +---+---+---+---+
95 | G | H | I | I | I | F |
96 +---+---+---+---+
97 | G | H | J | . | . | F |
98 +---+---+---+---+
99 | L | L | J | M | M | F |
100 +---+---+---+---+
101
102 =====
103 | | | | | STATISTICS
104 =====
105 Path length: 5 moves
106 Nodes visited: 560
107 Execution time: 541 ms
108 =====
Ln 13, Col 26 Spaces: 6 UTF-8 LF Plain Text

```

## 4.2. Test Case 1 - GBFS Heuristik Distance to Exit

Input : test/testing.txt

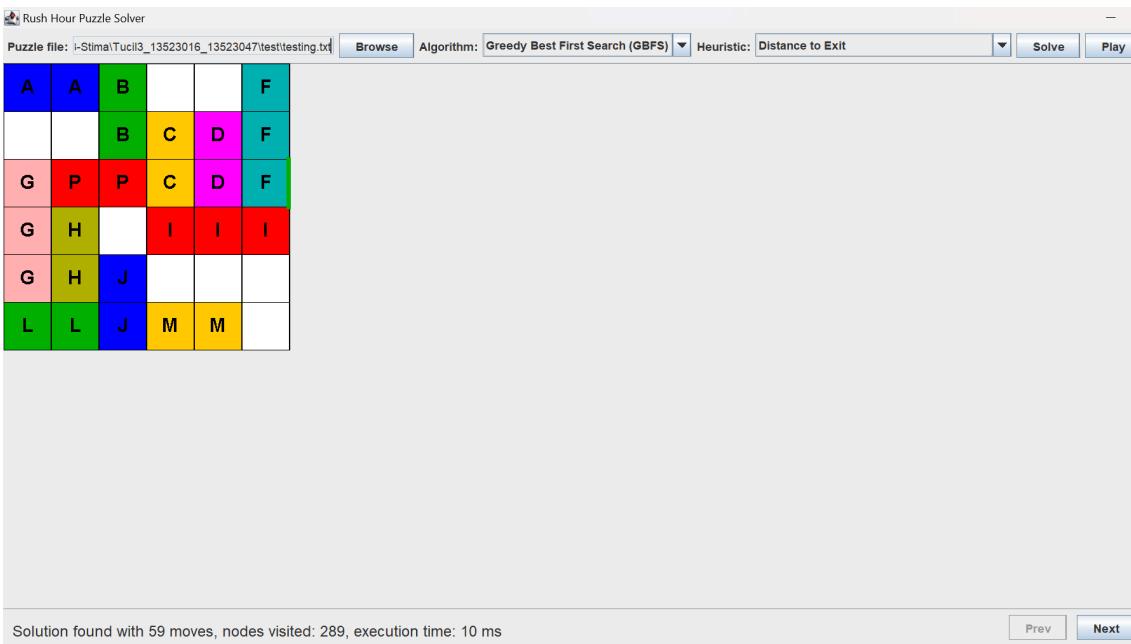
```

6 6
11
AAB..F
..BCDF
GPPCDFK

```

GH.III  
GHJ...  
LLJMM.

### Output GUI



### Output Simpan di File

```

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase1gbfs1.txt
TUCIL... D U
src
  algorithm
    gui
      BoardPanel.java
      ControlPanel.java
      MainFrame.java
      StatusPanel.java
    model
      Board.java
      Move.java
      Piece.java
      State.java
    utility
      Main.java
  test
    testcase1astar2... U
    testcase1astar3... U
    testcase1gbfs1... U
    testcase1ucs... U
    testcase2astar1... U
    testcase2astar3... U
    testcase2gbfs2... U
    testcase2ucs... U
    testcase3astar1... U
    testcase3astar3... U
    testcase3gbfs2... U
    testcase3ucs... U
    testcase4astar2... U
    testcase4astar3... U
    testcase4gbfs2... U
    testcase4ucs... U
outline timeline java projects
main* 0 0 Java Ready
Ln 1, Col 1 Spaces: 6 UTF-8 LF Plain Text

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase1gbfs1.txt
TUCIL... D U
src
  algorithm
    gui
      BoardPanel.java
      ControlPanel.java
      MainFrame.java
      StatusPanel.java
    model
      Board.java
      Move.java
      Piece.java
      State.java
    utility
      Main.java
  test
    testcase1astar2... U
    testcase1astar3... U
    testcase1gbfs1... U
    testcase1ucs... U
    testcase2astar1... U
    testcase2astar3... U
    testcase2gbfs2... U
    testcase2ucs... U
    testcase3astar1... U
    testcase3astar3... U
    testcase3gbfs2... U
    testcase3ucs... U
    testcase4astar2... U
    testcase4astar3... U
    testcase4gbfs2... U
    testcase4ucs... U
outline timeline java projects
main* 0 0 Java Ready
Ln 1, Col 1 Spaces: 6 UTF-8 LF Plain Text

```

The screenshots show the Eclipse IDE interface with two terminal windows. The top window displays the initial board state and a solution path containing 59 moves. The bottom window shows the final board state and statistics.

#### 4.3. Test Case 1 - A\* Heuristic Blocked Vehicles

Input : test/testing.txt

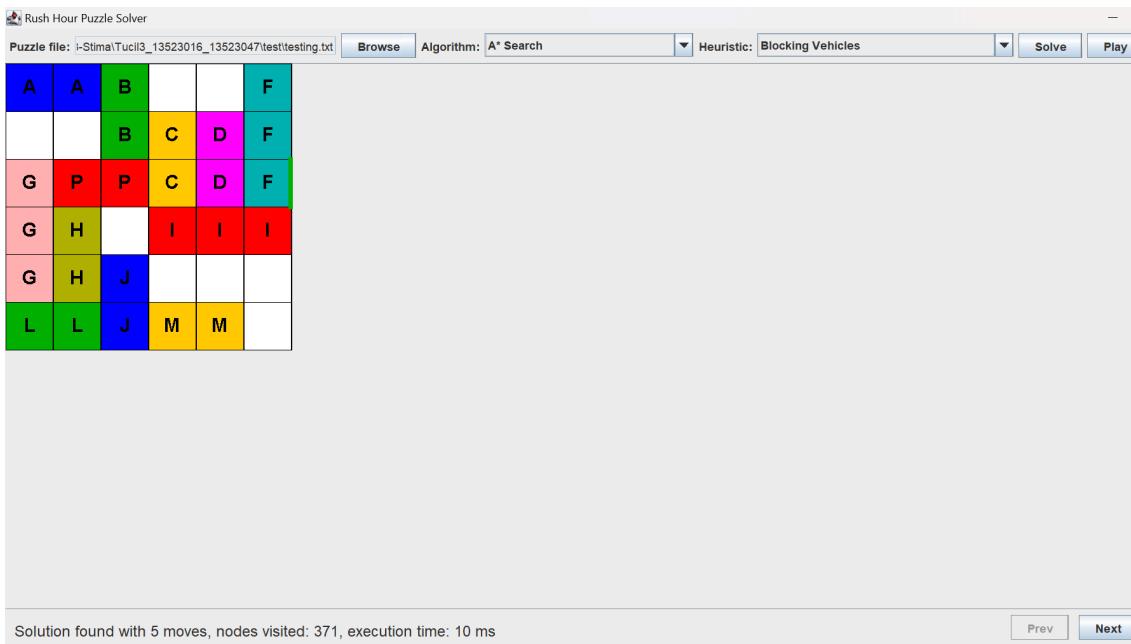
```

6 6
11
AAB..F
..BCDF
GPPCDFK

```

GH.III  
GHJ...  
LLJMM.

### Output GUI



### Output Simpan di File

```

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase1astar2.txt
TUCIL... D U
src
algorithm
gui
BoardPanel.java
ControlPanel.java
MainFrame.java
StatusPanel.java
model
Board.java
Move.java
Piece.java
State.java
utility
Main.java
test
testcase1astar2... U
testcase1astar3... U
testcase1ucs.txt U
testcase2astar1... U
testcase2astar3... U
testcase2gbfs2... U
testcase2ucs.txt U
Step by step solution:
Step 1: C-up
Step 5: P-right 3
Path length: 5 moves
Nodes visited: 371
Execution time: 10 ms
OUTLINE
TIMELINE
JAVA PROJECTS
Java Ready
Ln 1, Col 1 Spaces: 6 UTF-8 LF Plain Text

```

```

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase1astar2.txt
TUCIL... D U
src
algorithm
gui
BoardPanel.java
ControlPanel.java
MainFrame.java
StatusPanel.java
model
Board.java
Move.java
Piece.java
State.java
utility
Main.java
test
testcase1astar2... U
testcase1astar3... U
testcase1ucs.txt U
testcase2astar1... U
testcase2astar3... U
testcase2gbfs2... U
testcase2ucs.txt U
Step by step solution:
Step 1: C-up
Step 5: P-right 3
Path length: 5 moves
Nodes visited: 371
Execution time: 10 ms
OUTLINE
TIMELINE
JAVA PROJECTS
Java Ready
Ln 1, Col 1 Spaces: 6 UTF-8 LF Plain Text

```

#### 4.4. Test Case 1 - A\* Heuristik Combined

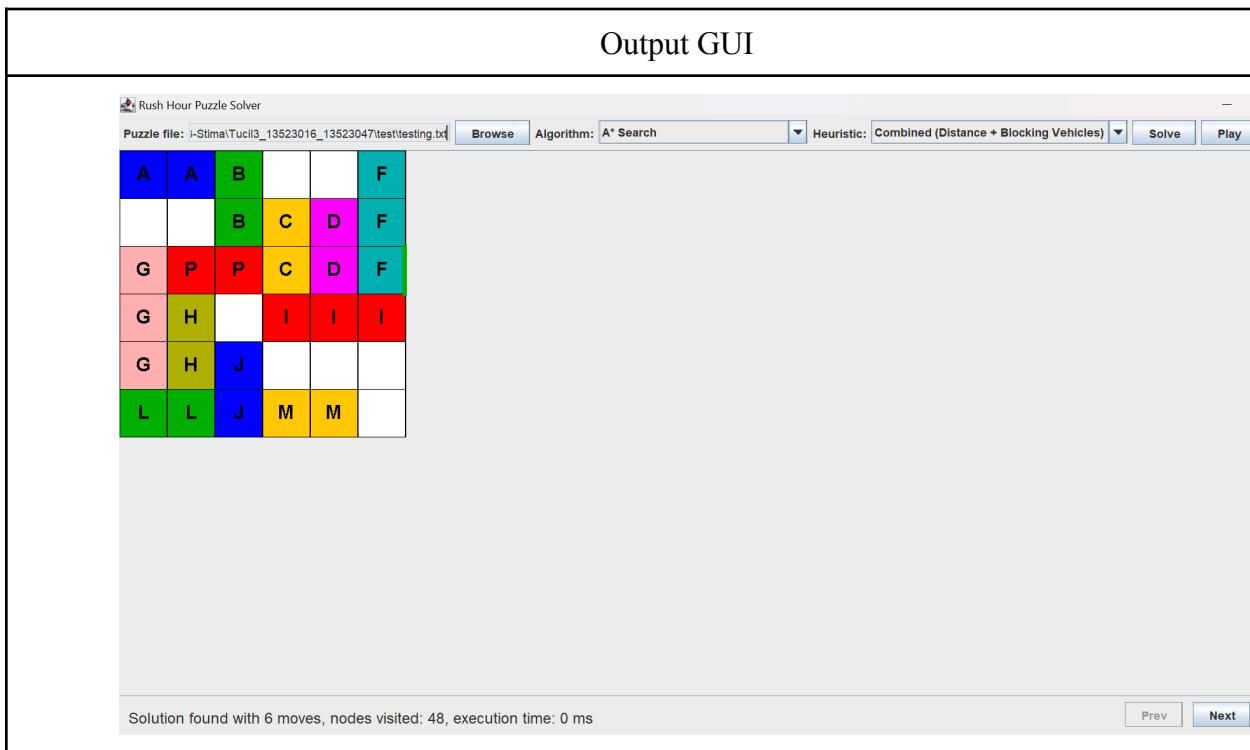
Input : test/testing.txt

```

6 6
11
AAB..F
..BCDF
GPPCDFK

```

GH.III  
GHJ...  
LLJMM.



Output Simpan di File

The image shows two side-by-side screenshots of a Java IDE (likely Eclipse or IntelliJ IDEA) displaying the output of an A\* search algorithm for Rush Hour puzzles.

**Top Window:**

- File Path:** test > testcase1astar3.txt
- Output:**

```

1  **** RUSH HOUR PUZZLE SOLVER ***
2
3 **** Initial Board:
4
5 +---+---+---+---+
6 | A | A | B | . | F |
7 +---+---+---+---+
8 +---+---+---+---+
9 | . | . | B | C | D | F |
10 +---+---+---+---+
11 | G | P | P | C | D | F |
12 +---+---+---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+---+
15 | G | H | J | . | . | . |
16 +---+---+---+---+
17 | L | L | J | M | M | . |
18 +---+---+---+---+
19
20 ===== SOLUTION FOUND =====
21 | | | | | | | | | | | |
22 =====
23 Solution path contains 6 moves:
24
25 Step by step solution:
26
27 Step 1: C-up
28 +---+---+---+---+
29 | A | A | B | C | F |
30 +---+---+---+---+
31 | . | . | B | C | D | F |
32 +---+---+---+---+
33 | G | P | P | . | D | F |
34 +---+---+---+---+
35 | G | H | . | I | I | I |
36 +---+---+---+---+

```

**Bottom Window:**

- File Path:** test > testcase1astar3.txt
- Output:**

```

94 +---+---+---+---+
95 | G | H | I | I | I | F |
96 +---+---+---+---+
97 | G | H | J | . | . | F |
98 +---+---+---+---+
99 | L | I | J | M | M | F |
100 +---+---+---+---+
101
102 Step 6: P-right
103 +---+---+---+---+
104 | A | A | B | C | D | . |
105 +---+---+---+---+
106 | . | . | B | C | D | . |
107 +---+---+---+---+
108 | G | . | . | . | P | P |
109 +---+---+---+---+
110 | G | H | I | I | I | F |
111 +---+---+---+---+
112 | G | H | J | . | . | F |
113 +---+---+---+---+
114 | L | I | J | M | M | F |
115 +---+---+---+---+
116
117 ===== STATISTICS =====
118 | | | | | | | | | | | |
119 =====
120 Path length: 6 moves
121 Nodes visited: 48
122 Execution time: 0 ms
123 =====

```

#### 4.5. Test Case 2 - UCS

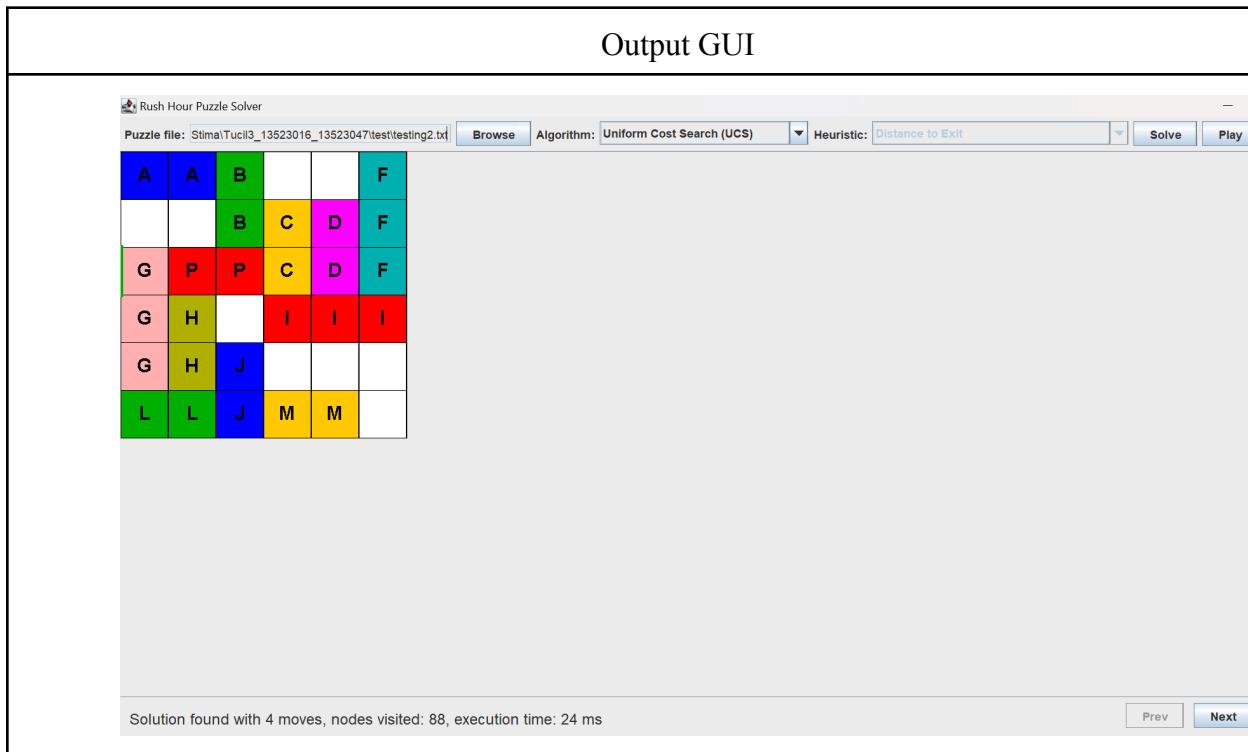
Input : test/testing2.txt

```

6 6
11
AAB..F
..BCDF

```

KGPPCDF  
GH.III  
GHJ...  
LLJMM.



Output Simpan di File

```

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase2ucs.txt
TUCIL... src ...
src ...
algorithm ...
gui ...
BoardPanel.java ...
ControlPanel.java ...
MainFrame.java ...
StatusPanel.java ...
model ...
Board.java ...
Move.java ...
Piece.java ...
State.java ...
utility ...
Main.java ...
test ...
testcase1ucs.txt ...
testcase2astar1... ...
testcase2astar3... ...
testcase2gbfs2... ...
testcase2ucs.txt ...
testcase3astar1... ...
testcase3astar3... ...
testcase3gbfs2... ...
testcase3ucs.txt ...
testcase4astar2... ...
testcase4astar3... ...
testcase4gbfs1... ...
testcase4gbfs2... ...
outline ...
timeline ...
java projects ...
main* ...
Java Ready ...
File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase2ucs.txt
TUCIL... src ...
src ...
algorithm ...
gui ...
BoardPanel.java ...
ControlPanel.java ...
MainFrame.java ...
StatusPanel.java ...
model ...
Board.java ...
Move.java ...
Piece.java ...
State.java ...
utility ...
Main.java ...
test ...
testcase1ucs.txt ...
testcase2astar1... ...
testcase2astar3... ...
testcase2gbfs2... ...
testcase2ucs.txt ...
testcase3astar1... ...
testcase3astar3... ...
testcase3gbfs2... ...
testcase3ucs.txt ...
testcase4astar2... ...
testcase4astar3... ...
testcase4gbfs1... ...
testcase4gbfs2... ...
outline ...
timeline ...
java projects ...
main* ...
Java Ready ...

```

#### 4.6. Test Case 2 - GBFS Heuristik Blocked Vehicles

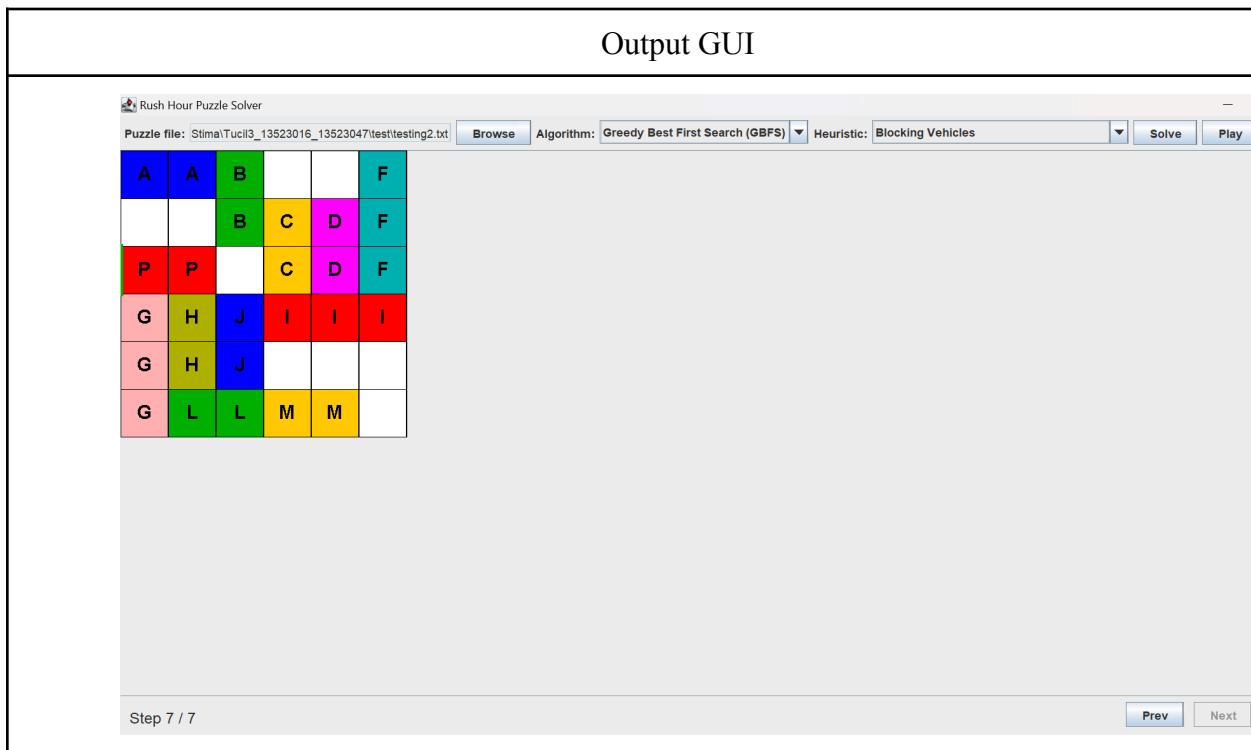
Input : test/testing2.txt

```

6 6
11
AAB..F
..BCDF
KGPPCDF

```

GH.III  
GHJ...  
LLJMM.



Output Simpan di File

```

File Edit Selection View ...
File Edit Selection View ...
 README.md testcase2gbfs2.txt README.md testcase2gbfs2.txt
test > testcase2gbfs2.txt
1 =====
2 | | *** RUSH HOUR PUZZLE SOLVER ***
3 =====
4
5 Initial Board:
6 +---+---+---+---+
7 | A | A | B | . | . | F |
8 +---+---+---+---+
9 | . | . | B | C | D | F |
10 +---+---+---+---+
11 | G | P | P | C | D | F |
12 +---+---+---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+---+
15 | G | H | J | . | . | .
16 +---+---+---+---+
17 | L | L | J | M | M | .
18 +---+---+---+---+
19
20 =====
21 | | | | | SOLUTION FOUND
22 =====
23 Solution path contains 7 moves:
24
25 Step by step solution:
26
27 Step 1: M-right
28 +---+---+---+---+
29 | A | A | B | . | . | F |
30 +---+---+---+---+
31 | . | . | B | C | D | F |
32 +---+---+---+---+
33 | G | P | P | C | D | F |
34 +---+---+---+---+
35 | G | H | . | I | I | I |
36 +---+---+---+---+
Ln 52, Col 26 Spaces: 2 UTF-8 LF {} Plain Tex

File Edit Selection View ...
File Edit Selection View ...
 README.md testcase2gbfs2.txt README.md testcase2gbfs2.txt
test > testcase2gbfs2.txt
107 +---+---+---+---+
108 | . | P | P | C | D | F |
109 +---+---+---+---+
110 | G | H | J | I | I | I |
111 +---+---+---+---+
112 | G | H | J | . | . | .
113 +---+---+---+---+
114 | G | L | L | M | M | .
115 +---+---+---+---+
116
117 Step 7: P-left
118 +---+---+---+---+
119 | A | A | B | . | . | F |
120 +---+---+---+---+
121 | . | . | B | C | D | F |
122 +---+---+---+---+
123 | P | P | . | C | D | F |
124 +---+---+---+---+
125 | G | H | J | I | I | I |
126 +---+---+---+---+
127 | G | H | J | . | . | .
128 +---+---+---+---+
129 | G | L | L | M | M | .
130 +---+---+---+---+
131
132 =====
133 | | | | | STATISTICS
134 =====
135 Path length: 7 moves
136 Nodes visited: 13
137 Execution time: 10 ms
138 =====
Ln 52, Col 26 Spaces: 2 UTF-8 LF {} Plain Tex

```

#### 4.7. Test Case 2 - A\* Heuristik Distance to Exit

Input : test/testing2.txt

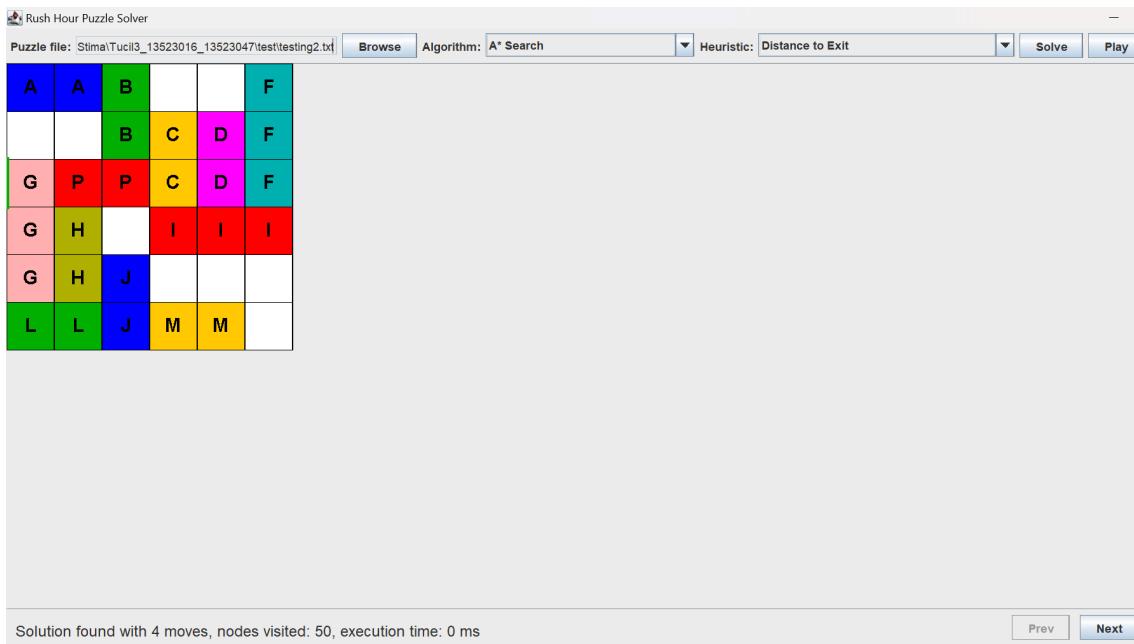
```

6 6
11
AAB..F
..BCDF
KGPPCDF

```

GH.III  
GHJ...  
LLJMM.

### Output GUI



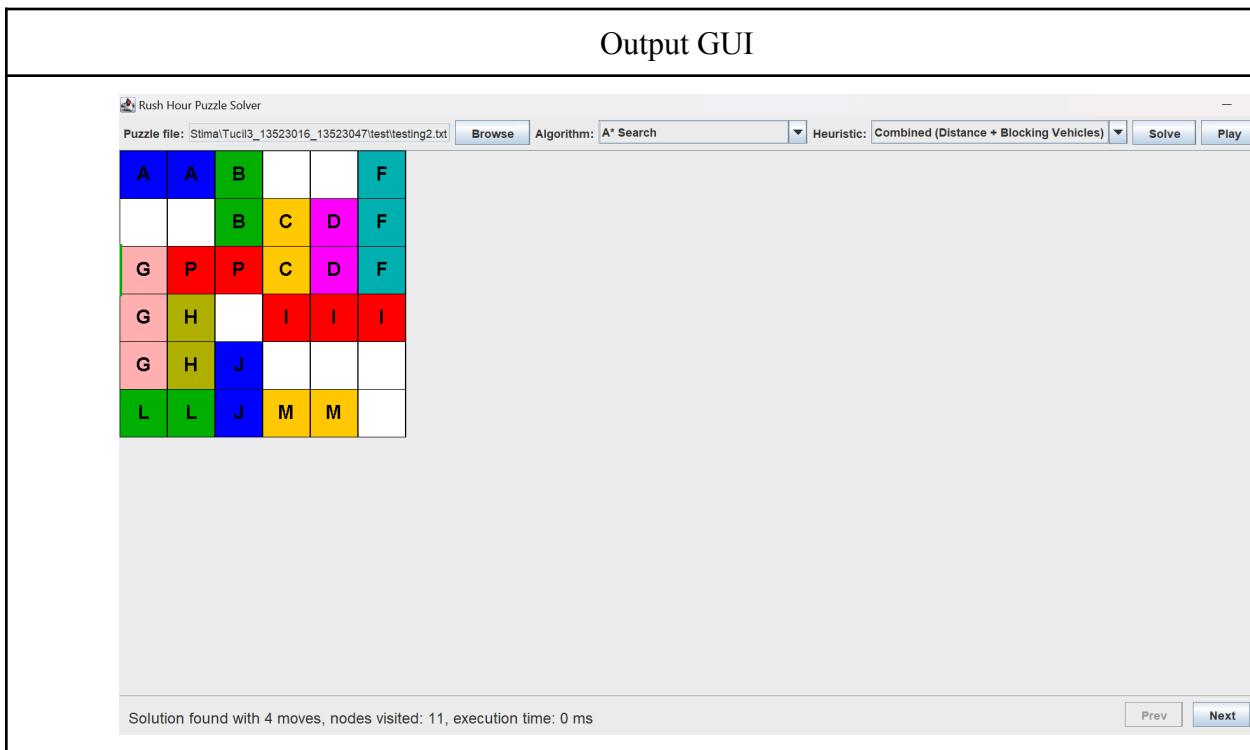
### Output Simpan di File

#### 4.8. Test Case 2 - A\* Heuristik Combined

Input : test/testing2.txt

66  
11  
AAB..F  
..BCDF  
KGPPCDE

GH.III  
GHJ...  
LLJMM.



Output Simpan di File

```

File Edit Selection View ... < > Tucil3_13523016_13523047 test > testcase2astar3.txt
EXPLORER ... testcase2astar3.txt
TUCIL3_13523016_13523047 test > testcase2astar3.txt
1 =====
2 | | *** RUSH HOUR PUZZLE SOLVER ***
3 =====
4
5 Initial Board:
6 +---+---+---+---+---+
7 | A | A | B | . | F |
8 +---+---+---+---+---+
9 | . | . | B | C | D | F |
10 +---+---+---+---+---+
11 | G | P | P | C | D | F |
12 +---+---+---+---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+---+---+
15 | G | H | J | . | . | . |
16 +---+---+---+---+---+
17 | L | L | J | M | K | . |
18 +---+---+---+---+---+
19
20 ===== SOLUTION FOUND =====
21 | | | | | |
22 =====
23 Solution path contains 4 moves:
24
25 Step by step solution:
26
27 Step 1: J-up
28 +---+---+---+---+---+
29 | A | A | B | . | F |
30 +---+---+---+---+---+
31 | . | . | B | C | D | F |
32 +---+---+---+---+---+
33 | G | P | P | C | D | F |
34 +---+---+---+---+---+
35 | G | H | J | I | I | I |
36 +---+---+---+---+---+

```

```

File Edit Selection View ... < > Tucil3_13523016_13523047 test > testcase2astar3.txt
EXPLORER ... testcase2astar3.txt
TUCIL3_13523016_13523047 test > testcase2astar3.txt
60 +---+---+---+---+---+
61 | . | . | B | C | D | F |
62 +---+---+---+---+---+
63 | . | P | P | C | D | F |
64 +---+---+---+---+---+
65 | G | H | J | I | I | I |
66 +---+---+---+---+---+
67 | G | H | J | . | . | . |
68 +---+---+---+---+---+
69 | G | L | L | M | M | . |
70 +---+---+---+---+---+
71
72 Step 4: P-left
73 +---+---+---+---+---+
74 | A | A | B | . | F |
75 +---+---+---+---+---+
76 | . | . | B | C | D | F |
77 +---+---+---+---+---+
78 | P | P | . | C | D | F |
79 +---+---+---+---+---+
80 | G | H | J | I | I | I |
81 +---+---+---+---+---+
82 | G | H | J | . | . | . |
83 +---+---+---+---+---+
84 | G | L | L | M | M | . |
85 +---+---+---+---+---+
86
87 =====
88 | | | | | | STATISTICS |
89 | | | | | |
90 Path length: 4 moves
91 Nodes visited: 11
92 Execution time: 0 ms
93 =====

```

#### 4.9. Test Case 3 - UCS

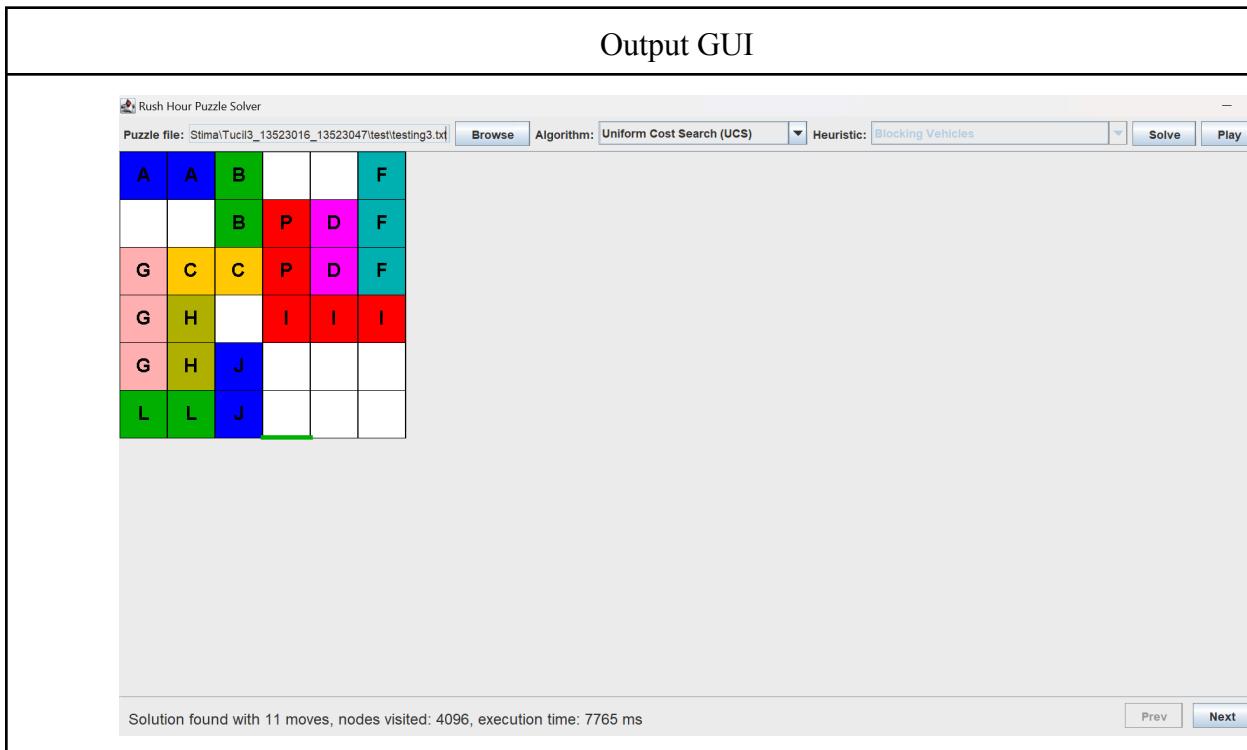
Input : test/testing3.txt

```

6 6
10
AAB..F
..BPDF
GCCPDF

```

GH.III  
GHJ...  
LLJ...  
K



Output Simpan di File

The image shows two side-by-side screenshots of a Java IDE (likely Eclipse or IntelliJ IDEA) displaying a Rush Hour puzzle solver project named "TUCIL".

**Top Window:**

- File Menu:** File, Edit, Selection, View, ...
- Toolbar:** Standard Java IDE toolbar.
- Search Bar:** Tucil3\_13523016\_13523047
- Left Panel (EXPLORER):** Shows the project structure under "TUCIL...".
- Right Panel (Code Editor):**

```

1  **** RUSH HOUR PUZZLE SOLVER ***
2
3  =====
4
5  Initial Board:
6 +---+---+---+---+---+
7 | A | A | B | . | F |
8 +---+---+---+---+---+
9 | . | . | B | P | D | F |
10 +---+---+---+---+---+
11 | G | C | C | P | . | F |
12 +---+---+---+---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+---+---+
15 | G | H | J | . | . | . |
16 +---+---+---+---+---+
17 | L | L | J | . | . | . |
18 +---+---+---+---+---+
19
20 ===== SOLUTION FOUND =====
21 | . | . | . | . | . | . |
22 =====
23 Solution path contains 11 moves:
24
25 Step by step solution:
26
27 Step 1: P-up
28 +---+---+---+---+---+
29 | A | A | B | P | . | F |
30 +---+---+---+---+---+
31 | . | . | B | P | D | F |
32 +---+---+---+---+---+
33 | G | C | C | . | D | F |
34 +---+---+---+---+---+
35 | G | H | . | I | I | I |
36 +---+---+---+---+---+

```
- Bottom Status Bar:** main\*, 0 0 0 Java Ready, Line 1, Col 1, Spaces: 6, UTF-8, LF, Plain Text.

**Bottom Window:**

- File Menu:** File, Edit, Selection, View, ...
- Toolbar:** Standard Java IDE toolbar.
- Search Bar:** Tucil3\_13523016\_13523047
- Left Panel (EXPLORER):** Shows the project structure under "TUCIL...".
- Right Panel (Code Editor):**

```

172 | . | . | J | . | . | F |
173 +---+---+---+---+---+
174 | L | L | J | . | . | F |
175 +---+---+---+---+---+
176 Step 11: P-down 4
177 +---+---+---+---+---+
178 | G | A | A | . | D | . |
179 +---+---+---+---+---+
180 | G | H | B | . | D | . |
181 +---+---+---+---+---+
182 | G | H | B | . | C | C |
183 +---+---+---+---+---+
184 | I | I | T | . | . | F |
185 +---+---+---+---+---+
186 | . | . | J | P | . | F |
187 +---+---+---+---+---+
188 | L | L | J | P | . | F |
189 +---+---+---+---+---+
190 | . | . | . | . | . | F |
191 +---+---+---+---+---+
192 =====
193 | . | . | . | . | . | . |
194 ===== STATISTICS =====
195 Path length: 11 moves
196 Nodes visited: 4096
197 Execution time: 7765 ms
198 =====

```
- Bottom Status Bar:** main\*, 0 0 0 Java Ready, Line 1, Col 1, Spaces: 6, UTF-8, LF, Plain Text.

#### 4.10. Test Case 3 - GBFS Heuristik Blocked Vehicles

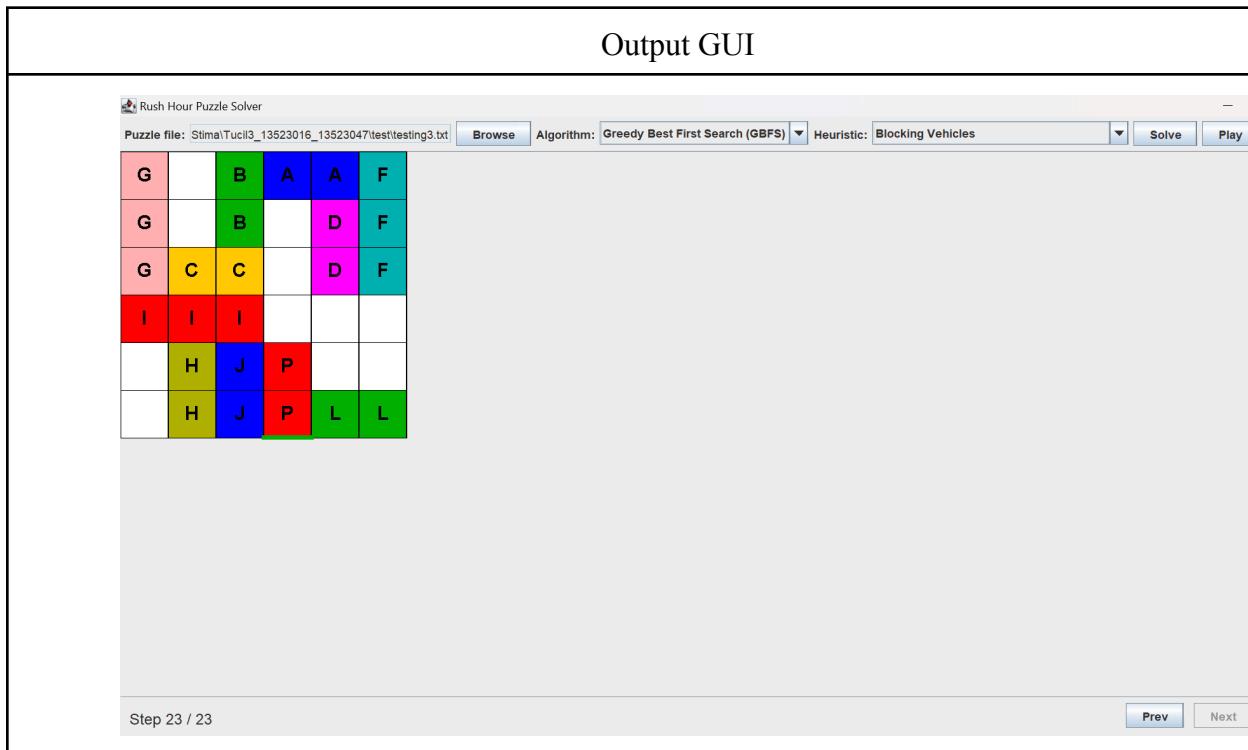
Input : test/testing3.txt

```

6 6
10
AAB..F
..BPDF
GCCPDF

```

GH.III  
GHJ...  
LLJ...  
K



Output Simpan di File

The image shows two side-by-side screenshots of the Eclipse IDE interface. Both windows have the title bar "Tucil3\_13523016\_13523047".

**Left Window (Screenshot 1):**

- File Menu:** File, Edit, Selection, View, ...
- Toolbar:** Back, Forward, Home, Search, etc.
- Search Bar:** Tucil3\_13523016\_13523047
- Left Panel (EXPLORER):**
  - TUCIL...
  - src
    - algorithm
    - gui
      - J BoardPanel.java
      - J ControlPanel.java
      - J MainFrame.java
      - J StatusPanel.java
    - model
      - J Board.java
      - J Move.java
      - J Piece.java
      - J State.java
    - > utility
    - J Main.java
  - test
    - E testcase1ucs.txt
    - E testcase2gbfs2... U
    - E testcase3astar1... U
    - E testcase3astar3... U
    - E testcase3gbfs2... U
    - E testcase4astar2... U
    - E testcase4astar3... U
    - E testcase4gbfs1... U
    - E testcase4gbfs2... U
    - E testcase4ucs.txt U
    - E testing.txt
    - E testing2.txt
    - E testing3.txt
- Bottom Status Bar:** main\* Java Ready

**Right Window (Screenshot 2):**

- File Menu:** File, Edit, Selection, View, ...
- Toolbar:** Back, Forward, Home, Search, etc.
- Search Bar:** Tucil3\_13523016\_13523047
- Left Panel (EXPLORER):** Same as the first window.
- Code Editor:**

```

344 | G | . | B | A | A | F |
345 +---+---+---+---+---+
346 | G | . | B | P | D | F |
347 +---+---+---+---+---+
348 | G | C | C | P | D | F |
349 +---+---+---+---+---+
350 | I | I | I | . | . | . |
351 +---+---+---+---+---+
352 | . | H | J | . | . | . |
353 +---+---+---+---+---+
354 | . | H | J | . | L | L |
355 +---+---+---+---+---+
356 Step 23: P-down 3
358 +---+---+---+---+---+
359 | G | . | B | A | A | F |
360 +---+---+---+---+---+
361 | G | . | B | . | D | F |
362 +---+---+---+---+---+
363 | G | C | C | . | D | F |
364 +---+---+---+---+---+
365 | I | I | I | . | . | . |
366 +---+---+---+---+---+
367 | . | H | J | P | . | . |
368 +---+---+---+---+---+
369 | . | H | J | P | L | L |
370 +---+---+---+---+---+
371 =====
372 | | | | STATISTICS |
373 | | | |
374 | | | =====
375 Path length: 23 moves
376 Nodes visited: 3152
377 Execution time: 120 ms
378 =====

```
- Bottom Status Bar:** main\* Java Ready

#### 4.11. Test Case 3 - A\* Heuristic Distance to Exit

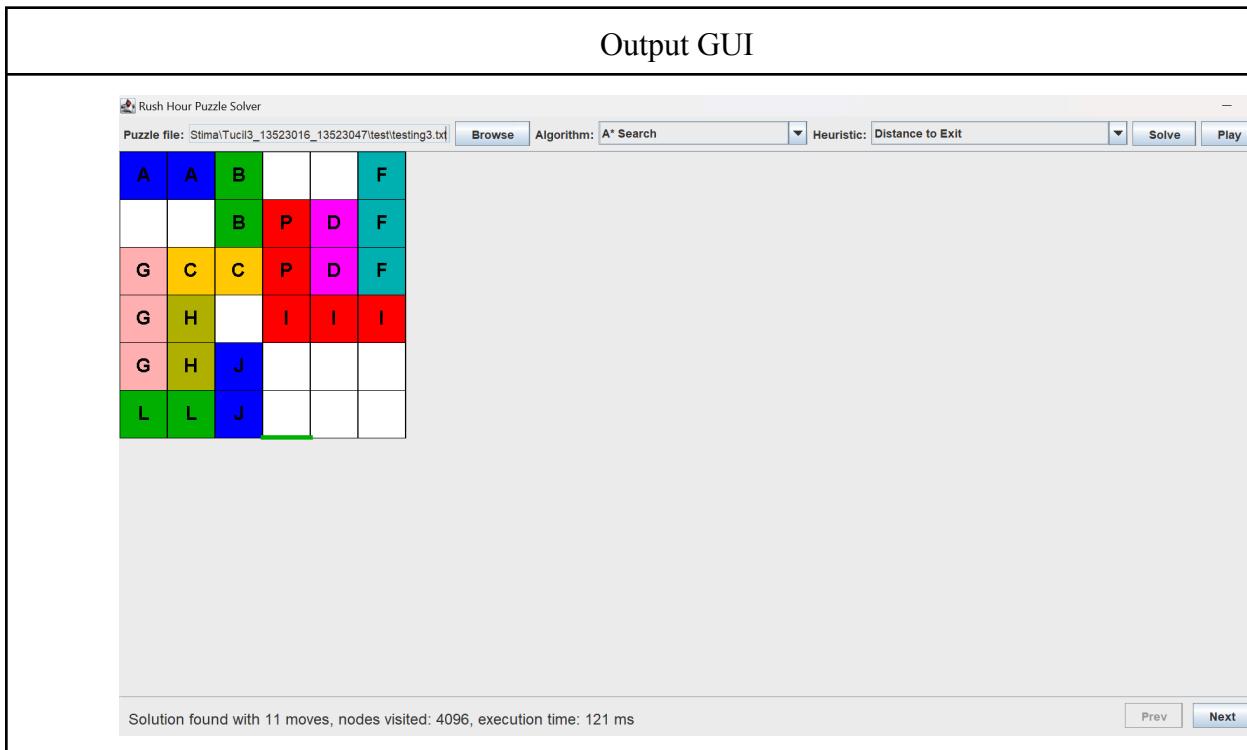
Input : test/testing3.txt

```

6 6
10
AAB..F
..BPDF
GCCPDF

```

GH.III  
GHJ...  
LLJ...  
K



Output Simpan di File

```

File Edit Selection View ...
File Edit Selection View ...
test > testcase3astar1.txt
test > testcase3astar1.txt
1 =====
2 | *** RUSH HOUR PUZZLE SOLVER ***
3 =====
4
5 Initial Board:
6 +---+---+---+---+
7 | A | A | B | . | . | F |
8 +---+---+---+---+
9 | . | . | B | P | D | F |
10 +---+---+---+---+
11 | G | C | C | P | D | F |
12 +---+---+---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+---+
15 | G | H | D | . | . | .
16 +---+---+---+---+
17 | L | L | I | . | . | .
18 +---+---+---+---+
19
20 =====
21 | | | SOLUTION FOUND
22 =====
23 Solution path contains 11 moves:
24
25 Step by step solution:
26
27 Step 1: P-up
28 +---+---+---+---+
29 | A | A | B | P | . | F |
30 +---+---+---+---+
31 | . | . | B | P | D | F |
32 +---+---+---+---+
33 | G | C | C | . | D | F |
34 +---+---+---+---+
35 | G | H | . | I | I | I |
36 +---+---+---+---+
Ln 49, Col 26 Spaces: 6 UTF-8 LF {} Plain Text

File Edit Selection View ...
File Edit Selection View ...
test > testcase3astar1.txt
test > testcase3astar1.txt
165 +---+---+---+---+
166 | G | H | B | P | D | . |
167 +---+---+---+---+
168 | G | H | B | . | C | C |
169 +---+---+---+---+
170 | I | I | I | . | . | F |
171 +---+---+---+---+
172 | . | . | J | . | . | F |
173 +---+---+---+---+
174 | L | L | J | . | . | F |
175 +---+---+---+---+
176
177 Step 11: P-down 4
178 +---+---+---+---+
179 | G | A | A | . | D | . |
180 +---+---+---+---+
181 | G | H | B | . | D | . |
182 +---+---+---+---+
183 | G | H | B | . | C | C |
184 +---+---+---+---+
185 | I | I | I | . | . | F |
186 +---+---+---+---+
187 | . | . | J | P | . | F |
188 +---+---+---+---+
189 | L | L | J | P | . | F |
190 +---+---+---+---+
191
192 =====
193 | | | STATISTICS
194 =====
195 Path length: 11 moves
196 Nodes visited: 4096
197 Execution time: 121 ms
198 =====
Ln 49, Col 26 Spaces: 6 UTF-8 LF {} Plain Text

```

#### 4.12. Test Case 3 - A\* Heuristik Combined

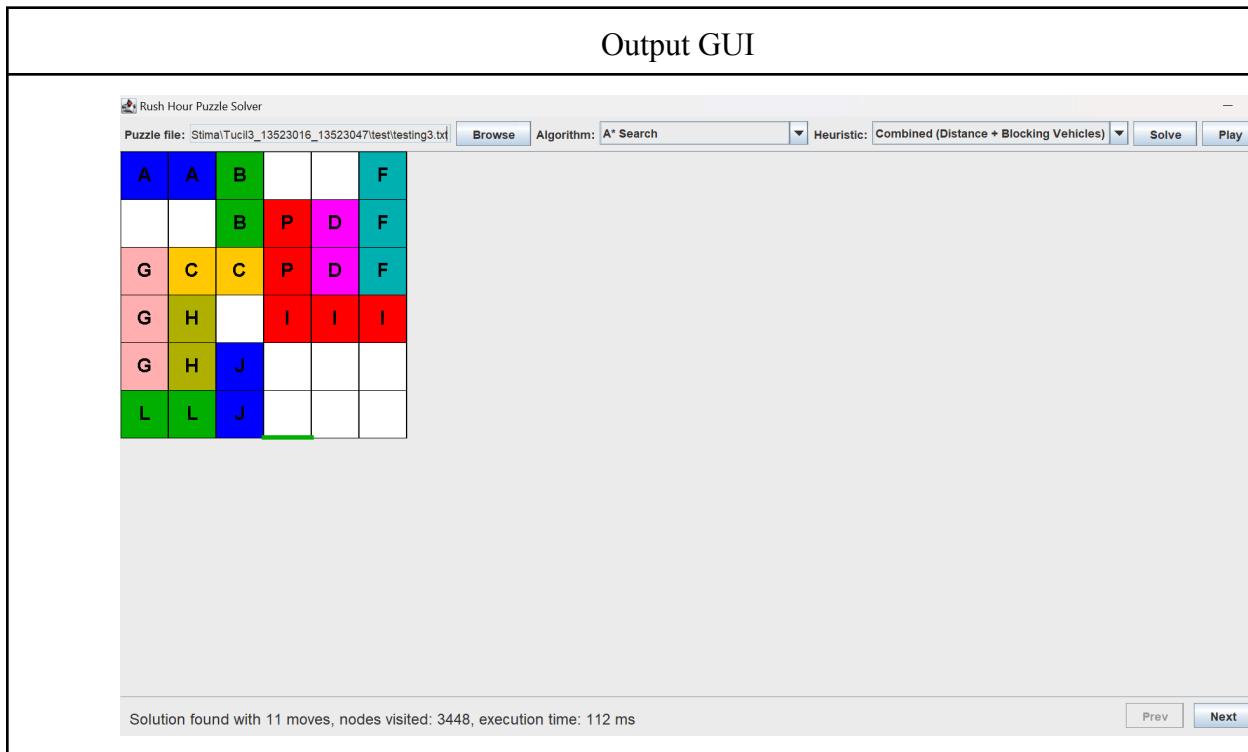
Input : test/testing3.txt

```

6 6
10
AAB..F
..BPDF
GCCPDF

```

GH.III  
GHJ...  
LLJ...  
K



Output Simpan di File

```

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase3astar3.txt
TUCIL... testcase3astar3.txt
src
algorithm
gui
BoardPanel.java
ControlPanel.java
MainFrame.java
StatusPanel.java
model
Board.java
Move.java
Piece.java
State.java
utility
Main.java
test
testcase1ucs.txt
testcase2gbfs2...
testcase3astar1...
testcase3astar3... U
testcase4astar2...
testcase4astar3...
testcase4gbfs1...
testcase4gbfs2...
testcase4ucs.txt
testing.txt
testing2.txt
testing3.txt
testing4.txt
Outline Timeline Java Projects
main* 0 0 Java Ready
Ln 1, Col 1 Spaces: 6 UTF-8 LF Plain Text

```

```

File Edit Selection View ...
File Edit Selection View ...
EXPLORER test > testcase3astar3.txt
TUCIL... testcase3astar3.txt
src
algorithm
gui
BoardPanel.java
ControlPanel.java
MainFrame.java
StatusPanel.java
model
Board.java
Move.java
Piece.java
State.java
utility
Main.java
test
testcase1ucs.txt
testcase2gbfs2...
testcase3astar1...
testcase3astar3... U
testcase4astar2...
testcase4astar3...
testcase4gbfs1...
testcase4gbfs2...
testcase4ucs.txt
testing.txt
testing2.txt
testing3.txt
testing4.txt
Outline Timeline Java Projects
main* 0 0 Java Ready
Ln 1, Col 1 Spaces: 6 UTF-8 LF Plain Text

```

The code in both editors is identical and represents a Java program for solving a Rush Hour puzzle using the A\* search algorithm. It includes imports for java.util.\* and javax.swing.\*. The main class is Main.java, which contains a static main method. The static main method reads input from a file named 'testing4.txt' located in the 'test' directory. The code then prints the initial board state and performs the search. The output shows the solution found and the step-by-step solution path.

#### 4.13. Test Case 4 - UCS

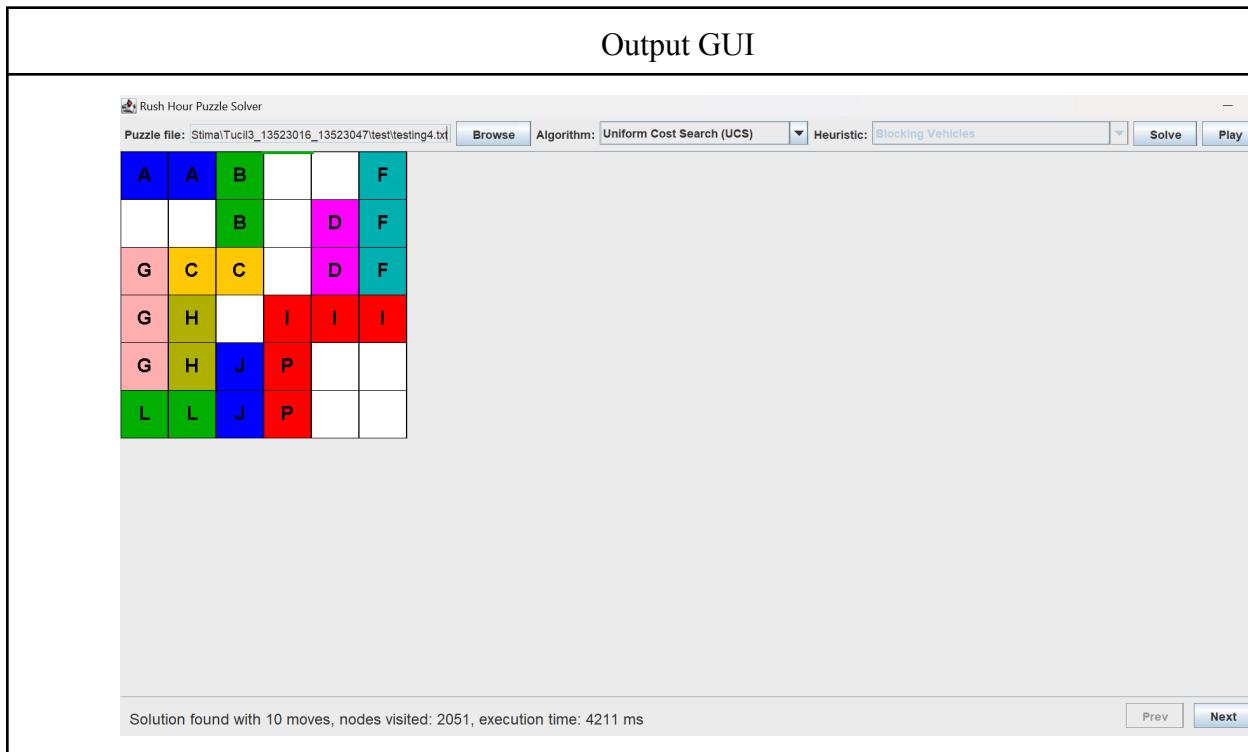
Input : test/testing4.txt

```

6 6
10
K
AAB..F
..B.DF

```

GCC.DF  
GH.III  
GHJP..  
LLJP..



Output Simpan di File

The image shows two side-by-side screenshots of a Java IDE (likely Eclipse or IntelliJ IDEA) running on a Windows operating system. Both windows have the title bar 'Tucil3\_13523016\_13523047'. The left window displays the solution path for a 6x6 Rush Hour puzzle with 10 cars, starting from state 'testing4.txt' and ending at state 'testcase4ucs.txt'. The right window shows the same process but with a different board configuration, likely a variation of the puzzle.

```

*** RUSH HOUR PUZZLE SOLVER ***
Initial Board:
+---+---+---+---+
| A | A | B | . | F |
+---+---+---+---+
| . | . | B | . | D | F |
+---+---+---+---+
| G | C | C | . | D | F |
+---+---+---+---+
| G | H | . | I | I | I |
+---+---+---+---+
| G | H | J | P | . | . |
+---+---+---+---+
| L | L | J | P | . | . |
+---+---+---+---+
| L | L | J | P | . | . |
+---+---+---+---+
SOLUTION FOUND
Solution path contains 10 moves:
Step by step solution:
Step 1: D-up
+---+---+---+---+
| A | A | B | . | F |
+---+---+---+---+
| . | . | B | . | D | F |
+---+---+---+---+
| G | C | C | . | F |
+---+---+---+---+
| G | H | . | I | I | I |
+---+---+---+---+
| G | H | J | P | . | . |
+---+---+---+---+
| L | L | J | P | . | . |
+---+---+---+---+
| L | L | J | P | . | . |
+---+---+---+---+
Step 10: P-up 4
+---+---+---+---+
| G | A | A | D | . |
+---+---+---+---+
| G | H | B | P | D | . |
+---+---+---+---+
| G | H | B | . | C | C |
+---+---+---+---+
| I | I | I | . | . | F |
+---+---+---+---+
| I | I | J | . | . | F |
+---+---+---+---+
| L | L | J | . | . | F |
+---+---+---+---+
| L | L | J | . | . | F |
+---+---+---+---+
STATISTICS
Path length: 10 moves
Nodes visited: 2051
Execution time: 4211 ms

```

#### 4.14. Test Case 4 - GBFS Heuristik Distance to Exit

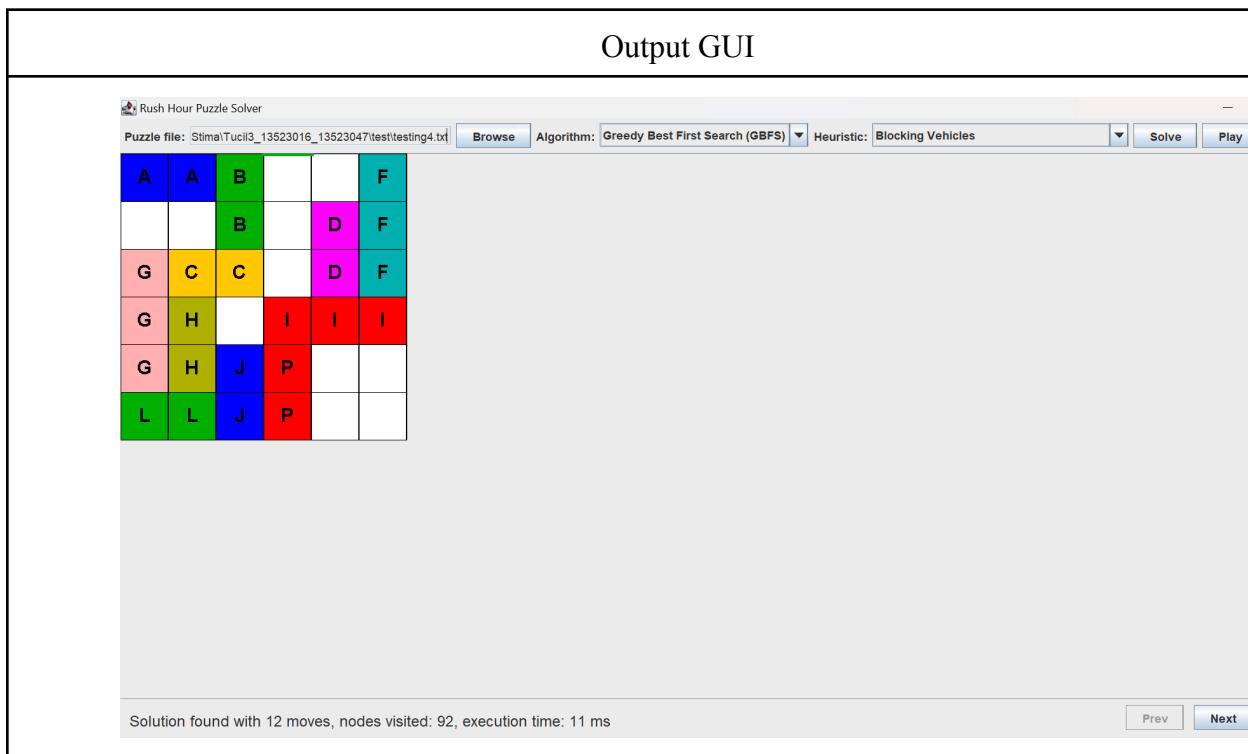
Input : test/testing4.txt

```

6 6
10
K
AAB..F
..B.DF

```

GCC.DF  
GH.III  
GHJP..  
LLJP..



Output Simpan di File

```

1 *** RUSH HOUR PUZZLE SOLVER ***
2
3 =====
4 Initial Board:
5 +---+---+---+ +---+---+
6 | A | A | B | . | F |
7 +---+---+---+ +---+---+
8 | . | . | B | . | D | F |
9 +---+---+---+ +---+---+
10 | . | . | B | . | D | F |
11 | G | C | C | . | D | F |
12 +---+---+---+ +---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+ +---+---+
15 | G | H | J | P | . | . |
16 +---+---+---+ +---+---+
17 | L | L | J | P | . | . |
18 +---+---+---+ +---+---+
19
20 ===== SOLUTION FOUND =====
21 | . | . | . | . | . | . |
22 =====
23 Solution path contains 12 moves:
24
25 Step by step solution:
26
27 Step 1: I-left
28 +---+---+---+ +---+---+
29 | A | A | B | . | F |
30 +---+---+---+ +---+---+
31 | . | . | B | . | D | F |
32 +---+---+---+ +---+---+
33 | G | C | C | . | D | F |
34 +---+---+---+ +---+---+
35 | 6 | H | I | I | I | I |
36 +---+---+---+ +---+---+

```

```

185 | I | I | I | P | . | F |
186 +---+---+---+ +---+---+
187 | . | . | J | . | F |
188 +---+---+---+ +---+---+
189 | L | L | J | . | F |
190 +---+---+---+ +---+---+
191 Step 12: P-up 2
192
193 +---+---+---+ +---+---+
194 | G | A | A | P | D | . |
195 +---+---+---+ +---+---+
196 | G | H | B | P | D | . |
197 +---+---+---+ +---+---+
198 | G | H | B | . | C | C |
199 +---+---+---+ +---+---+
200 | I | I | I | . | . | F |
201 +---+---+---+ +---+---+
202 | . | . | J | . | . | F |
203 +---+---+---+ +---+---+
204 | L | L | J | . | F |
205 +---+---+---+ +---+---+
206
207 =====
208 | . | . | . | . | . | . |
209 ===== STATISTICS =====
210 Path length: 12 moves
211 Nodes visited: 92
212 Execution time: 11 ms
213 =====

```

#### 4.15. Test Case 4 - A\* Heuristik Blocked Vehicles

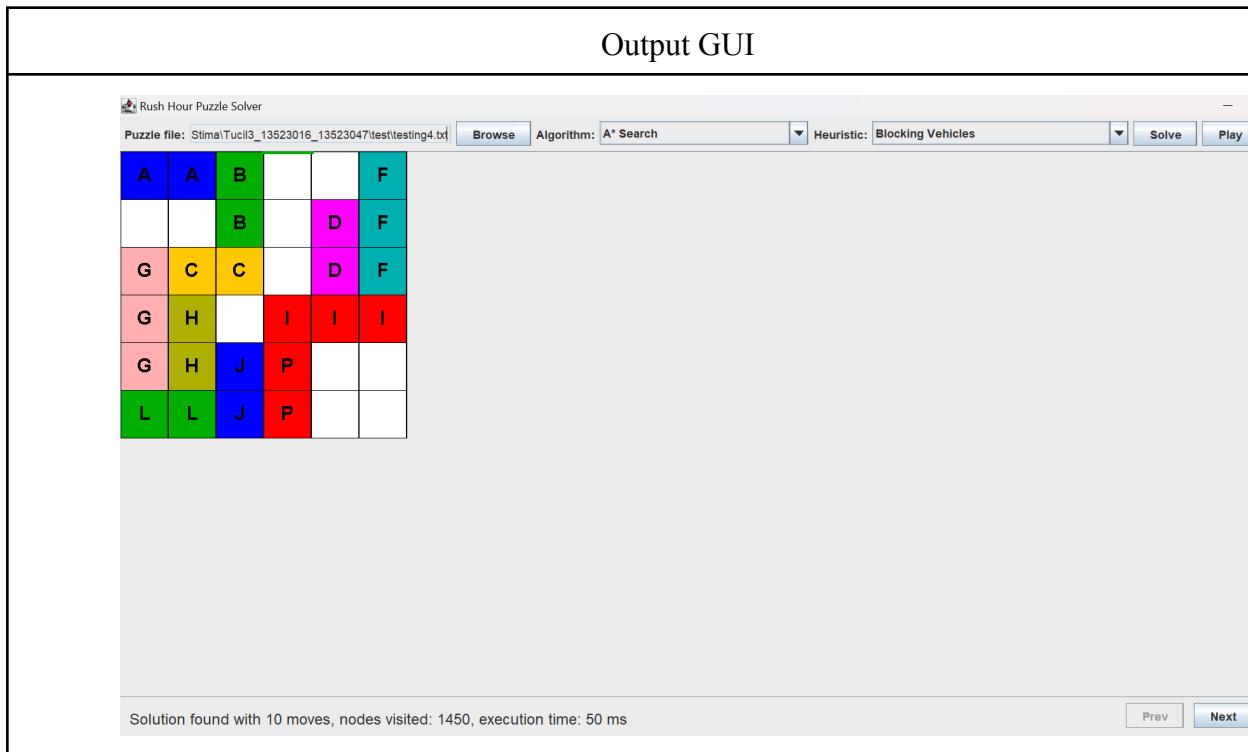
Input : test/testing4.txt

```

6 6
10
K
AAB..F
..B.DF

```

GCC.DF  
GH.III  
GHJP..  
LLJP..



Output Simpan di File

```

1 =====
2 | *** RUSH HOUR PUZZLE SOLVER ***
3 =====
4
5 Initial Board:
6 +---+---+---+ +---+---+
7 | A | A | B | . | . | F |
8 +---+---+---+ +---+---+
9 | . | . | B | . | D | F |
10 +---+---+---+ +---+---+
11 | G | C | C | . | D | F |
12 +---+---+---+ +---+---+
13 | G | H | . | I | I | I |
14 +---+---+---+ +---+---+
15 | G | H | J | P | . | . |
16 +---+---+---+ +---+---+
17 | L | L | J | P | . | . |
18 +---+---+---+ +---+---+
19
20 ===== SOLUTION FOUND =====
21 | . | . | . | . | . | .
22 =====
23 Solution path contains 10 moves:
24
25 Step by step solution:
26
27 Step 1: D-up
28 +---+---+---+ +---+---+
29 | A | A | B | . | D | F |
30 +---+---+---+ +---+---+
31 | . | . | B | . | D | F |
32 +---+---+---+ +---+---+
33 | G | C | C | . | . | F |
34 +---+---+---+ +---+---+
35 | G | H | . | I | I | I |
36 +---+---+---+ +---+---+

```

```

150 +---+---+---+ +---+---+
151 | G | H | B | . | D | . |
152 +---+---+---+ +---+---+
153 | G | H | B | . | C | C |
154 +---+---+---+ +---+---+
155 | I | I | I | I | . | F |
156 +---+---+---+ +---+---+
157 | . | . | J | P | . | F |
158 +---+---+---+ +---+---+
159 | L | L | J | P | . | F |
160 +---+---+---+ +---+---+
161
162 Step 10: P-up 4
163 +---+---+---+ +---+---+
164 | G | A | A | P | D | . |
165 +---+---+---+ +---+---+
166 | G | H | B | P | D | . |
167 +---+---+---+ +---+---+
168 | G | H | B | . | C | C |
169 +---+---+---+ +---+---+
170 | I | I | I | T | . | F |
171 +---+---+---+ +---+---+
172 | . | . | J | . | . | F |
173 +---+---+---+ +---+---+
174 | L | L | J | P | . | F |
175 +---+---+---+ +---+---+
176
177 =====
178 | . | . | . | . | . | .
179 ===== STATISTICS =====
180 Path length: 10 moves
181 Nodes visited: 1450
182 Execution time: 50 ms
183 =====

```

#### 4.16. Test Case 4 - A\* Heuristik Combined

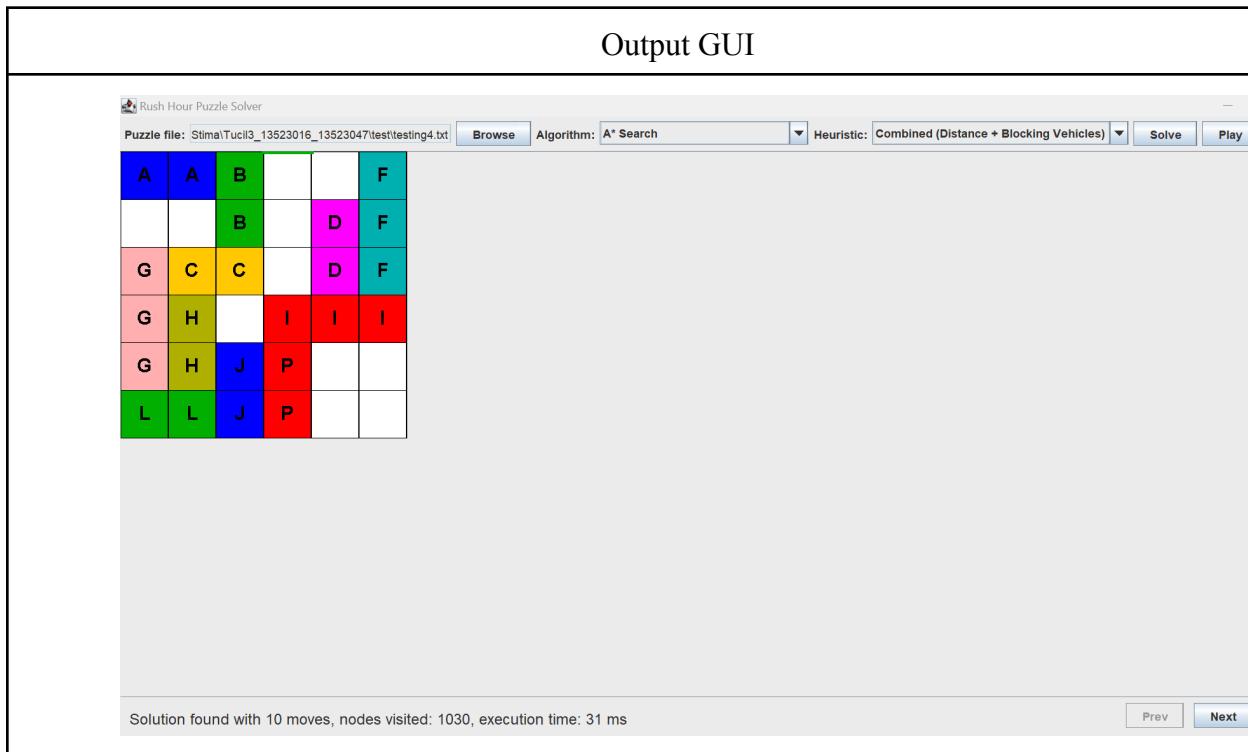
Input : test/testing4.txt

```

6 6
10
K
AAB..F
..B.DF

```

GCC.DF  
GH.III  
GHJP..  
LLJP..



Output Simpan di File

```
File Edit Selection View ... ← → ⌂ Tuci3_13523016_13523047

test > testcase4astar3.txt
1 =====
2 | *** RUSH HOUR PUZZLE SOLVER ***
3 =====
4
5 Initial Board:
6 +-----+-----+
7 | A | A | B | . | . | F |
8 +-----+-----+
9 | . | . | B | . | D | F |
10 +-----+-----+
11 | G | C | C | . | D | F |
12 +-----+-----+
13 | G | H | . | I | I | I |
14 +-----+-----+
15 | G | H | J | P | . | .
16 +-----+-----+
17 | L | L | J | P | . | .
18 +-----+-----+
19
20 =====
21 | | | | SOLUTION FOUND
22 =====
23 Solution path contains 10 moves:
24
25 Step by step solution:
26
27 Step 1: D-up
28 +-----+
29 | A | A | B | . | D | F |
30 +-----+
31 | . | . | B | . | D | F |
32 +-----+
33 | G | C | C | . | . | F |
34 +-----+
35 | G | H | . | I | I | I |
36 +-----+
37
38 main* ⌂ ⌂ 0 Δ 0 ⌂ Java: Ready ⌂ Ln 75, Col 26 Spaces: 6 UTF-8 ⌂ Plain Text

File Edit Selection View ... ← → ⌂ Tuci3_13523016_13523047
test > testcase4astar3.txt
152 +-----+
153 | G | H | B | . | C | C |
154 +-----+
155 | I | I | I | . | . | F |
156 +-----+
157 | . | . | J | P | . | F |
158 +-----+
159 | L | L | J | P | . | F |
160 +-----+
161
162 Step 10: P-up 4
163 +-----+
164 | G | A | A | P | D | .
165 +-----+
166 | G | H | B | P | D | .
167 +-----+
168 | G | H | B | . | C | C |
169 +-----+
170 | I | I | I | . | . | F |
171 +-----+
172 | . | . | J | P | . | F |
173 +-----+
174 | L | L | J | P | . | F |
175 +-----+
176
177 =====
178 | | | | STATISTICS
179 =====
180 Path length: 10 moves
181 Nodes visited: 1030
182 Execution time: 31 ms
183 =====
```

#### **4.17. Analisis Percobaan Algoritma Pathfinding**

Tiga algoritma pencarian yang dibandingkan adalah Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A\*. UCS memiliki keunggulan utama dalam selalu menemukan solusi yang optimal, yaitu dengan jumlah langkah minimum. Namun, kelemahannya terletak pada waktu eksekusi yang sangat lama serta jumlah node yang dikunjungi paling banyak, terutama pada test case yang kompleks. Di sisi lain, GBFS menunjukkan performa terbaik dalam hal kecepatan eksekusi dan jumlah node yang dikunjungi. Sayangnya, solusi yang dihasilkannya seringkali tidak optimal karena terlalu

mengandalkan estimasi heuristik tanpa mempertimbangkan jarak tempuh sebenarnya. A\* hadir sebagai kompromi yang seimbang antara UCS dan GBFS. Algoritma ini mampu menemukan solusi optimal seperti UCS, namun dengan efisiensi waktu dan jumlah node yang jauh lebih baik. Meskipun tidak secepat GBFS, A\* lebih andal dalam menghasilkan solusi yang optimal.

Tiga heuristik yang digunakan dalam eksperimen ini adalah Distance to Exit, Blocking Vehicles, dan Combined (Distance + Blocking Vehicles). Heuristik Distance to Exit menunjukkan performa sedang dalam hal waktu eksekusi dan jumlah node yang dikunjungi. Ia cukup efektif untuk menyelesaikan puzzle sederhana, namun kurang mampu mengatasi kasus kompleks. Heuristik Blocking Vehicles memperlihatkan kinerja yang lebih baik dibanding Distance to Exit karena mampu mendeteksi kemacetan yang disebabkan oleh kendaraan penghalang. Heuristik gabungan (Combined) terbukti menjadi yang paling unggul. Ia secara konsisten menunjukkan performa terbaik, baik dari sisi waktu eksekusi maupun jumlah node yang dikunjungi. Bahkan pada algoritma GBFS yang biasanya menghasilkan solusi suboptimal, heuristik Combined mampu memberikan solusi yang lebih mendekati optimal dibandingkan dua heuristik lainnya.

Dalam hal efisiensi dan kualitas solusi, terdapat trade-off yang jelas antara ketiga algoritma. UCS, meskipun menjamin solusi optimal, memerlukan biaya komputasi tertinggi. Sebaliknya, GBFS sangat efisien secara waktu namun menghasilkan solusi dengan panjang langkah yang tidak optimal. A\* menjadi pilihan terbaik di antara keduanya, karena ia mampu menemukan solusi optimal seperti UCS, tetapi dengan waktu eksekusi dan efisiensi yang jauh lebih baik. Penggunaan heuristik Combined pada A\* semakin memperkuat performanya, menjadikannya metode paling seimbang dan efektif pada sebagian besar skenario yang diuji.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Implementasi dan analisis algoritma Uniform Cost Search (UCS), Greedy Best First Search, dan A\* dalam penyelesaian masalah Rush Hour menunjukkan perbedaan signifikan dalam hal efisiensi dan optimalisasi solusi. UCS yang hanya mempertimbangkan biaya jalur sejauh ini memberikan solusi optimal namun dengan eksplorasi simpul yang lebih banyak. Greedy Best First Search lebih cepat karena mengandalkan heuristik saja, tetapi tidak menjamin solusi optimal. Sementara itu, A\* menggabungkan kelebihan kedua algoritma dengan memanfaatkan fungsi evaluasi  $f(n)=g(n)+h(n)$   $f(n) = g(n) + h(n)$ , sehingga dapat menemukan solusi optimal secara lebih efisien dengan bantuan heuristik admissible. Hasil ini menegaskan pentingnya pemilihan heuristik yang tepat untuk mempercepat pencarian tanpa mengorbankan kualitas solusi. Secara

keseluruhan, A\* menjadi pilihan terbaik dalam menyelesaikan masalah pencarian jalur pada Rush Hour dengan keseimbangan antara kecepatan dan optimalitas.

### **5.2. Saran**

Dalam pengembangan selanjutnya, program ini dapat ditingkatkan terutama pada heuristik yang dipakai dalam algoritma A\* agar lebih akurat dan sesuai dengan karakteristik masalah Rush Hour, sehingga ruang pencarian bisa dipersempit dan efisiensi algoritma meningkat. Pengembangan tampilan program dalam bentuk antarmuka grafis (GUI) dapat mempermudah pengguna dalam mengoperasikan aplikasi serta memahami proses pencarian secara lebih intuitif. Selain itu, penambahan fitur visualisasi yang menampilkan jalur solusi dan proses pencarian secara interaktif akan sangat membantu dalam menganalisis hasil. Untuk mendukung keandalan program, pengujian lebih lanjut pada berbagai kondisi masalah dan skenario permainan juga perlu dilakukan agar performa algoritma tetap optimal di berbagai situasi.

Tautan repository Github [https://github.com/ClarissaNT44/Tucil3\\_13523016\\_13523047.git](https://github.com/ClarissaNT44/Tucil3_13523016_13523047.git)

## DAFTAR PUSTAKA

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

[2][https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf)

## LAMPIRAN

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	✓	