IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour

Jawaban Tugas 18/5/2025

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada
Semester 2 Tahun Akademik 2024/2025

Disusun oleh:

Rhio Bimo Prakoso S	(13523123)
Frederiko Eldad M	(13523147)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUSI TEKNOLOGI BANDUNG

2024

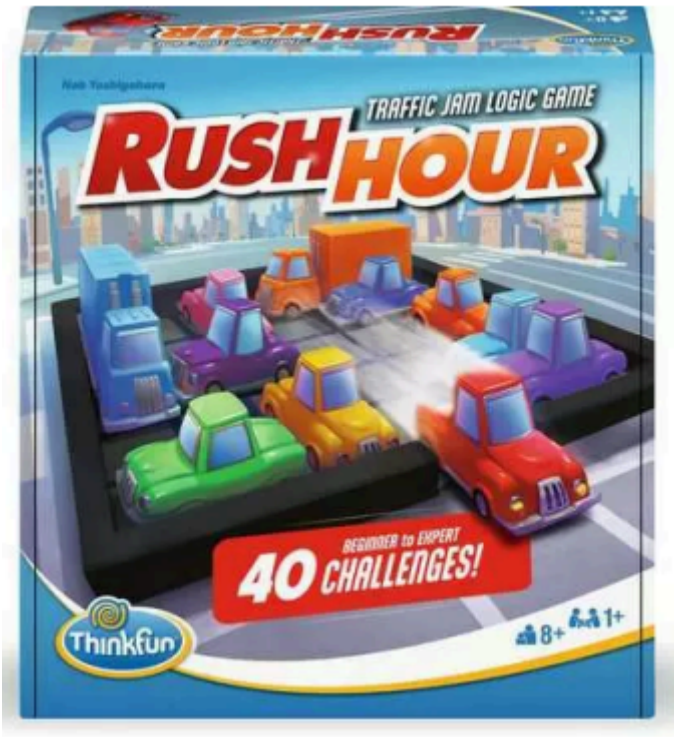
DAFTAR ISI

DAFTAR ISI **2**
DAFTAR GAMBAR **3**
BAB I
DESKRIPSI MASALAH **4**
BAB II
TEORI SINGKAT **6**
2.1. Uniform Cost Search (UCS) 6
2.2. Greedy Best First Search (GBFS) 6
2.3 A-star Algorithm (A-star) 7
BAB III
METODE PENYELESAIAN **8**
3.1. Heuristics 8
3.2. Mapping Heuristics dengan Algoritma Pathfinding 8
3.3. Optimization 8
BAB IV
IMPLEMENTASI **9**
4.1. Implementasi Program 9
4.2. Kode Sumber 9
4.2.1. Model 9
4.2.2. Searching 17
4.2.3. Heuristic 24
4.2.4. Checker 34
4.2.5. Exception 35
4.2.6. GUI 36
4.2.7. Main.java 50
BAB V
IMPLEMENTASI BONUS **51**
5.1. Graphical User Interface (GUI) 51
5.2. Heuristik Alternatif 52
5.3. Pathfinding Alternatif 60
BAB VI
EKSPERIMEN DAN ANALISIS **63**
6.1. Pengujian Program 63
6.2. Analisis Hasil Percobaan 64
6.2.1. Analisis Kompleksitas Algoritma UCS 64
6.2.2. Analisis Kompleksitas Algoritma GBFS 64
6.2.3. Analisis Kompleksitas Algoritma A* (A-star) 65
LAMPIRAN **66**
Checklist Spesifikasi Program 66
Pembagian Tugas 66
Repository 66
DAFTAR PUSTAKA **68**

DAFTAR GAMBAR

Gambar 1.1. Rush Hour Puzzle 5
Gambar 5.1. GUI; Main Menu **52**

BAB I
DESKRIPSI MASALAH



Gambar 1.1. *Rush Hour Puzzle*
(Sumber:https://www.thinkfun.com/en-US/products/educational-games/rush-hour-76582)

Rush Hour adalah sebuah permainan puzzle logika berbasis grid yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6x6) agar mobil utama (biasanya berwarna merah) dapat keluar dari kemacetan melalui pintu keluar di sisi papan. Setiap kendaraan hanya bisa bergerak lurus ke depan atau ke belakang sesuai dengan orientasinya (horizontal atau vertikal), dan tidak dapat berputar. Tujuan utama dari permainan ini adalah memindahkan mobil merah ke pintu keluar dengan jumlah langkah seminimal mungkin.

Komponen penting dari permainan Rush Hour terdiri dari:

1. Papan – Papan merupakan tempat permainan dimainkan. Papan terdiri atas cell, yaitu sebuah singular point dari papan. Sebuah piece akan menempati cell-cell pada papan. Ketika permainan dimulai, semua piece telah diletakkan di dalam papan dengan konfigurasi tertentu berupa lokasi piece dan orientasi, antara horizontal atau vertikal. Hanya primary piece yang dapat digerakkan keluar papan melewati pintu keluar. Piece yang bukan primary piece tidak dapat digerakkan keluar papan. Papan memiliki satu pintu keluar yang pasti berada di dinding papan dan sejajar dengan orientasi primary piece.

2. Piece – Piece adalah sebuah kendaraan di dalam papan. Setiap piece memiliki posisi, ukuran, dan orientasi. Orientasi sebuah piece hanya dapat berupa horizontal atau vertikal–tidak mungkin diagonal. Piece dapat memiliki beragam ukuran, yaitu jumlah cell yang ditempati oleh piece. Secara standar, variasi ukuran sebuah piece adalah 2-piece (menempati 2 cell) atau 3-piece (menempati 3 cell). Suatu piece tidak dapat digerakkan melewati/menembus piece yang lain.
3. Primary Piece – Primary piece adalah kendaraan utama yang harus dikeluarkan dari papan (biasanya berwarna merah). Hanya boleh terdapat satu primary piece.
4. Pintu Keluar – Pintu keluar adalah tempat primary piece dapat digerakkan keluar untuk menyelesaikan permainan
5. Gerakan — Gerakan yang dimaksudkan adalah pergeseran piece di dalam permainan. Piece hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu piece tidak dapat digerakkan melewati/menembus piece yang lain.

Alur program:

1. **[INPUT]** Konfigurasi permainan/test case dalam format .txt yang berisi:
    a. Dimensi Papan (A x B)
    b. Banyak piece yang **BUKAN** *primary piece* (N)
    c. Konfigurasi papan dengan pintu keluar dilambangkan **X** dan *primary piece* dilambangkan **P**. Cell kosong dilambangkan dengan titik (*period*) '**.**'
    Contoh file txt:

```
6 6
12
A A B . . F
. . B C D F
G P P C D F K
G H . I I I
G H J . . .
L L J M M .
```

2. **[INPUT]** algoritma pathfinding yang digunakan
3. **[INPUT]** heuristic yang digunakan (bonus)
4. **[OUTPUT]** Benyaknya **gerakan** yang diperiksa (banyak 'node' yang dikunjungi)
5. **[OUTPUT]** Waktu eksekusi program
6. **[OUTPUT]** Konfigurasi papan pada setiap tahap pergerakan. ('animasi')

BAB II
TEORI SINGKAT

2.1. *Uniform Cost Search (UCS)*

Uniform Cost Search merupakan algoritme penelusuran graf yang mencari jalur ber‑biaya total paling kecil dari simpul awal ke simpul tujuan. Termasuk golongan uninformed search, UCS tidak memakai informasi jarak ke goal; ia hanya memprioritaskan jalur dengan akumulasi biaya terendah sejauh ini.

Algoritme ini mengevaluasi setiap simpul n dengan fungsi:

$$f(n) = g(n)$$

di mana *g(n)* adalah biaya dari start hingga *n*. Simpul dengan nilai *f(n)* paling kecil selalu diambil dari *priority queue* untuk diperluas. Jika kemudian ditemukan rute yang lebih murah menuju simpul yang sama, rute yang lebih mahal dibuang.

Selama semua bobot langkah bernilai non-negatif, UCS menjamin solusi optimal. Metode ini cocok untuk persoalan di mana biaya minimum lebih penting daripada kecepatan, misalnya penentuan rute, penjadwalan, atau puzzle berbobot. Namun, tanpa bantuan heuristik, UCS dapat menjadi lambat karena harus menelusuri seluruh jalur termurah satu per satu sebelum mencoba jalur lain.

2.2. *Greedy Best First Search (GBFS)*

Greedy Best-First Search adalah algoritme *informed search* yang mengandalkan heuristik untuk “menyerbu” ke arah goal secepat mungkin. Pada setiap langkah ia memilih simpul dengan estimasi jarak ke tujuan terkecil, tanpa memedulikan biaya yang telah dikeluarkan untuk mencapainya.

Fungsi evaluasinya adalah:

$$f(n) = h(n)$$

di mana h(n) merupakan taksiran jarak dari n ke goal. Strategi yang hanya melihat ke depan ini bisa sangat cepat jika heuristiknya akurat, tetapi karena mengabaikan biaya lampau, jalur yang dipilih tidak dijamin paling murah. GBFS mudah terperangkap pada optimum lokal atau jalur buntu yang tampak dekat secara heuristik tetapi mahal secara nyata.

Algoritma ini berguna ketika diperlukan respons cepat dan optimalitas bukan prioritas utama, misalnya navigasi awal, keputusan sementara, atau aplikasi real-time dengan batas waktu ketat.

2.3 A-star *Algorithm* (A-star)

A* Search menggabungkan kekuatan UCS dan GBFS dengan menyeimbangkan biaya nyata dan estimasi sisa biaya. Setiap simpul dievaluasi menggunakan:

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

dengan g(n) = biaya dari start ke n dan h(n) = perkiraan biaya dari n ke goal.

Dengan selalu memperluas simpul ber-f(n) terendah, A* memprioritaskan jalur yang sejauh ini murah sekaligus menjanjikan. Apabila heuristik h(n) admissible (tidak melebih-lebihkan) dan konsisten, A* pasti menemukan jalur optimal.

Keunggulannya adalah mampu mencapai solusi terbaik lebih efisien daripada UCS karena ruang pencarian disempitkan oleh heuristik. Meski demikian, pada ruang pencarian sangat besar atau heuristik kurang akurat, kebutuhan memori dan waktu A* tetap bisa tinggi.

BAB III
METODE PENYELESAIAN

Permainan Rush Hour dapat dimodelkan sebagai penelusuran pada graf ruang-keadaan:

- Simpul : mewakili satu konfigurasi papan (posisi semua kendaraan).
- Sisi : mewakili satu gerakan legal, yaitu menggeser sebuah mobil searah orientasinya.

Dengan pemodelan ini, algoritma pencarian jalur seperti Uniform Cost Search (UCS), Greedy Best-First Search (GBFS), dan A* bisa digunakan untuk menemukan solusi.

Target penelusuran adalah keadaan akhir, konfigurasi di mana mobil utama berhasil keluar. Solusi sendiri berupa rangkaian keadaan dari posisi awal hingga akhir, yang sekaligus menunjukkan urutan langkah penyelesaian. Untuk lebih detailnya sebagai berikut:

3.1. *Heuristics*

Heuristik utama yang dipakai adalah…

3.2. *Mapping Heuristics* dengan Algoritma Pathfinding

…

3.3. *Optimization*

…

BAB IV
IMPLEMENTASI

4.1. Implementasi Program

Implementasi program dibagi menjadi beberapa sumber kode. Masing-masing sumber kode memiliki peran khusus untuk penyelesaian masalah kompresi gambar. Pembagian sumber kode dilakukan untuk menyederhanakan masalah agar dapat lebih mudah dipahami. Berikut adalah kode sumber yang ada dalam implementasi program.

4.2. Kode Sumber

4.2.1. Model

4.2.1.1. Board.java

```
package com.java.model;

import com.java.exception.MoveBlockedException;
import java.util.*;

public class Board {
   private final int rows, cols;
   private final char[][] grid;
   private final Map<Character, Piece> pieces = new HashMap<>();
   private int exitRow = Integer.MIN_VALUE, exitCol = Integer.MIN_VALUE;

   public Board(int rows, int cols) {
      this.rows = rows;
      this.cols = cols;
      grid = new char[rows][cols];
      for (int r = 0; r < rows; r++)
         Arrays.fill(grid[r], '.');
   }

   public void setExit(int r, int c) {
      exitRow = r;
      exitCol = c;
   }

   public void parseCell(int r, int c, char ch) {
      if (ch == 'K') {
         setExit(r, c);
      } else if (Character.isLetter(ch)) {
         grid[r][c] = ch;
         if (!pieces.containsKey(ch)) {
            pieces.put(ch, new Piece(ch, r, c, 1, true, ch == 'P'));
         } else {
            Piece p = pieces.get(ch);
            int newSize   = p.getSize() + 1;
            boolean horiz = (r == p.getRow());
            int anchorR   = horiz ? p.getRow() : Math.min(p.getRow(), r);
```

```
            int anchorC   = horiz ? Math.min(p.getCol(), c) : p.getCol();
            p.setHorizontal(horiz);
            p.setSize(newSize);
            p.setPosition(anchorR, anchorC);
        }
    }
}

public char getCell(int r, int c)    { return grid[r][c]; }
public int getExitRow()              { return exitRow; }
public int getExitCol()              { return exitCol; }
public int getRows()                 { return rows; }
public int getCols()                 { return cols; }
public Map<Character, Piece> getPieces() { return pieces; }

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();

    final String[] COLORS = {
        "\u001B[32m", // GREEN (0)
        "\u001B[33m", // YELLOW (1)
        "\u001B[34m", // BLUE (2)
        "\u001B[35m", // PURPLE (3)
        "\u001B[36m", // CYAN (4)
        "\u001B[90m", // BRIGHT_BLACK (5)
        "\u001B[92m", // BRIGHT_GREEN (6)
        "\u001B[93m", // BRIGHT_YELLOW (7)
        "\u001B[94m", // BRIGHT_BLUE (8)
        "\u001B[95m", // BRIGHT_PURPLE (9)
        "\u001B[96m", // BRIGHT_CYAN (10)
        "\u001B[97m", // BRIGHT_WHITE (11)
        "\u001B[30m"  // BLACK (12)
    };

    final String RESET = "\u001B[0m";
    final String WHITE = "\u001B[37m";
    final String RED = "\u001B[31m";


sb.append(WHITE).append("+").append("-".repeat(this.getCols())).append("+").append(RES
ET).append("\n");

    for (int r = 0; r < this.getRows(); r++) {
        sb.append(WHITE).append("|").append(RESET);
        for (int c = 0; c < this.getCols(); c++) {
            char cell = this.getCell(r, c);

            if (cell == '.') {
                sb.append(WHITE).append(cell).append(RESET);
            } else if (cell == 'P') {
                sb.append(RED).append(cell).append(RESET);
            } else {
                int colorIndex = cell % COLORS.length;
                sb.append(COLORS[colorIndex]).append(cell).append(RESET);
            }
```

```
            }
            sb.append(WHITE).append("|").append(RESET).append("\n");
        }


sb.append(WHITE).append("+").append("-".repeat(this.getCols())).append("+").append(RES
ET);

        sb.append("\n").append(WHITE).append("Exit at: (")
          .append(this.getExitRow()).append(", ").append(this.getExitCol()).append(")")
          .append(RESET);

        return sb.toString();
    }


    public String toPlainString() {
        StringBuilder sb = new StringBuilder();

        sb.append("+");
        for (int c = 0; c < this.getCols(); c++) {
            sb.append("-");
        }
        sb.append("+\n");

        for (int r = 0; r < this.getRows(); r++) {
            sb.append("|");
            for (int c = 0; c < this.getCols(); c++) {
                sb.append(this.getCell(r, c));
            }
            sb.append("|\n");
        }

        sb.append("+");
        for (int c = 0; c < this.getCols(); c++) {
            sb.append("-");
        }
        sb.append("+");

        sb.append("\nExit at: (")
          .append(exitRow).append(", ").append(exitCol).append(")");

        return sb.toString();
    }

    public void printBoard() { System.out.println(this.toString()); }

    public void movePiece(Piece p, int x, int y) throws MoveBlockedException {
        if (p.getSize() > 1) {
            if (p.isHorizontal() && y != 0)
                throw new IllegalArgumentException(
                  "Piece '" + p.getId() + "' is horizontal; cannot move vertically");
            if (p.isVertical() && x != 0)
                throw new IllegalArgumentException(
                  "Piece '" + p.getId() + "' is vertical; cannot move horizontally");
        }
```

```
        int nr = p.getRow() - y;
        int nc = p.getCol() + x;
        boolean horiz = p.isHorizontal();
        int sz      = p.getSize();

        List<int[]> dest = new ArrayList<>();
        for (int i = 0; i < sz; i++) {
            int r = nr + (horiz ? 0 : i);
            int c = nc + (horiz ? i : 0);
            if (r < 0 || r >= rows || c < 0 || c >= cols) {
                System.out.println("\nMove out of bounds: (" + r + ", " + c + ")");
                printBoard();
                throw new MoveBlockedException(null);
            }
            char occ = grid[r][c];
            if (occ != '.' && occ != p.getId()) {
                System.out.println("\nMove blocked by piece: " + occ);
                printBoard();
                throw new MoveBlockedException(pieces.get(occ));
            }
            dest.add(new int[]{r, c});
        }

        for (int[] cell : p.occupiedCells()) {
            grid[cell[0]][cell[1]] = '.';
        }

        p.setPosition(nr, nc);

        for (int[] cell : dest) {
            grid[cell[0]][cell[1]] = p.getId();
        }
    }
}
```

4.2.1.2. Config.java

```
package com.java.model;

import com.java.exception.InvalidConfigurationException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class Config {
    public static Board loadConfig(String filename)
      throws InvalidConfigurationException
    {
        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String[] dims = br.readLine().trim().split("\\s+");
            int A = Integer.parseInt(dims[0]);       // rows
            int B = Integer.parseInt(dims[1]);       // cols
            int declaredCount = Integer.parseInt(br.readLine().trim());
```

```
Board board = new Board(A, B);

List<String> lines = new ArrayList<>();
String ln;
while ((ln = br.readLine()) != null) {
   lines.add(ln);
}

int gridRowsSeen = 0;
boolean exitFound = false;

for (String line : lines) {
   int xIdx = line.indexOf('K');

   if (!exitFound
      && gridRowsSeen < A
      && xIdx >= 0
      && line.length() == B + 1
      && (xIdx == 0 || xIdx == B))
   {
      int row = gridRowsSeen;
      int col = xIdx == 0 ? -1 : B;
      board.setExit(row, col);
      if (xIdx == 0) {
         for (int c = 0; c < B; c++) {
            board.parseCell(row, c, line.charAt(c + 1));
         }
      } else {
         for (int c = 0; c < B; c++) {
            board.parseCell(row, c, line.charAt(c));
         }
      }
      gridRowsSeen++;
      exitFound = true;

   } else if (xIdx >= 0 && line.trim().equals("K") && !exitFound) {
      int row = gridRowsSeen == 0 ? -1 : A;
      board.setExit(row, line.indexOf('K'));
      exitFound = true;

   } else if (gridRowsSeen < A) {
      if (line.length() < B) {
         throw new InvalidConfigurationException(
           "Grid row " + gridRowsSeen + " must have ≥ " + B + " chars"
         );
      }
      for (int c = 0; c < B; c++) {
         board.parseCell(gridRowsSeen, c, line.charAt(c));
      }
      gridRowsSeen++;
   }
}

if (gridRowsSeen < A) {
   throw new InvalidConfigurationException(
```

```
                "Expected " + A + " grid rows, but only found " + gridRowsSeen
            );
        }
        if (!exitFound) {
            throw new InvalidConfigurationException("No exit 'K' found in configuration");
        }

        Map<Character, Piece> pcs = board.getPieces();
        if (!pcs.containsKey('P')) {
            throw new InvalidConfigurationException("No primary piece 'P' found");
        }
        if (pcs.size() - 1 != declaredCount) {
            throw new InvalidConfigurationException(
                "Declared piece count mismatch: expected "
                + declaredCount + " non-primary pieces, found "
                + (pcs.size() - 1)
            );
        }

        int er = board.getExitRow(), ec = board.getExitCol();
        boolean exitLeft   = ec < 0;
        boolean exitRight  = ec > B - 1;
        boolean exitTop    = er < 0;
        boolean exitBottom = er > A - 1;

        for (Piece p : pcs.values()) {
            if (p.getSize() == 1) {
                p.setHorizontal(exitLeft || exitRight);
            }
        }

        return board;

    } catch (IOException e) {
            throw new InvalidConfigurationException("Error reading configuration: " +
e.getMessage());
    } catch (NumberFormatException e) {
            throw new InvalidConfigurationException("Invalid number format: " +
e.getMessage());
    }
  }
}
```

4.2.1.3. Piece.java

```
package com.java.model;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Piece {
  public static int piece_count = 0;
  private final char id;
  private int row, col;
```

```
private int size;
private boolean horizontal;
private final boolean primary;

public boolean isVertical() {
   return !horizontal;
}

public Piece(char id, int row, int col, int size, boolean horizontal, boolean primary) {
   this.id        = id;
   this.row       = row;
   this.col       = col;
   this.size      = size;
   this.horizontal = horizontal;
   this.primary    = primary;
   piece_count++;
}

public char getId()       { return id; }
public int  getRow()      { return row; }
public int  getCol()      { return col; }
public int  getSize()     { return size; }
public boolean isHorizontal() { return horizontal; }
public boolean isPrimary()    { return primary; }

void setPosition(int row, int col) {
   this.row = row;
   this.col = col;
}
void setSize(int size) {
   this.size = size;
}
void setHorizontal(boolean horizontal) {
   this.horizontal = horizontal;
}

public List<int[]> occupiedCells() {
   List<int[]> cells = new ArrayList<>();
   for (int i = 0; i < size; i++) {
      int r = row + (horizontal ? 0 : i);
      int c = col + (horizontal ? i : 0);
      cells.add(new int[]{r, c});
   }
   return cells;
}

@Override
public String toString() {
   return "Piece{id=" + id +
         ", row=" + row + ", col=" + col +
         ", size=" + size +
         ", horiz=" + horizontal +
         ", primary=" + primary + '}';
}

@Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Piece)) return false;
        Piece p = (Piece) o;
        return id == p.id &&
                row == p.row &&
                col == p.col &&
                size == p.size &&
                horizontal == p.horizontal &&
                primary == p.primary;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, row, col, size, horizontal, primary);
    }
}
```

4.2.1.4. Save.java

```
package com.java.model;

import com.java.model.Board;
import com.java.searching.Move;
import com.java.searching.SolverResult;
import com.java.searching.State;
import com.java.searching.heuristic.HeuristicType;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

public class Save {
    public static void saveSolution(SolverResult result, State initialState, String filename,
                                    String originalPuzzle, String algorithmName, String heuristic)
            throws IOException {
        String timestamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
        String fullFilename = filename + "_" + timestamp + ".txt";

        // mkdir like ahh type shi
        Files.createDirectories(Paths.get("test/solutions"));

        try (BufferedWriter writer = new BufferedWriter(new FileWriter("test/solutions/" +
fullFilename))) {
            writer.write("Rush Hour Solution\n");
            writer.write("==================\n\n");
            writer.write("Original puzzle: " + originalPuzzle + "\n");
            writer.write("Algorithm: " + algorithmName + "\n");
            writer.write("Heuristic: " + heuristic + "\n");
```

```
            writer.write("Number of moves: " + result.path.size() + "\n");
            writer.write("Nodes created: " + result.nodesCreated + "\n");
            writer.write("Execution time: " + result.timeMs + " ms\n\n");

            writer.write("Solution path:\n");
            writer.write("-------------\n\n");

            writer.write("Initial state:\n");
            writer.write(initialState.getBoard().toPlainString());
            writer.write("\n\n");

            State currentState = initialState;
            for (int i = 0; i < result.path.size(); i++) {
                Move move = result.path.get(i);
                currentState = move.next;

                writer.write("Step " + (i + 1) + ":\n");
                writer.write("Move piece '" + move.id + "' ");
                if (move.dr < 0) writer.write("left " + Math.abs(move.dr));
                if (move.dr > 0) writer.write("right " + move.dr);
                if (move.dc < 0) writer.write("down " + Math.abs(move.dc));
                if (move.dc > 0) writer.write("up " + move.dc);
                writer.write("\n\n");

                writer.write(currentState.getBoard().toPlainString());
                writer.write("\n\n");

                if (i < result.path.size() - 1) {
                    writer.write("↓\n\n");
                }
            }
        }
    }

    // Add an overloaded method that takes a HeuristicType:
    public static void saveSolution(SolverResult result, State initialState, String filename,
                                    String originalPuzzle, String algorithmName, HeuristicType
heuristicType)
            throws IOException {
        saveSolution(result, initialState, filename, originalPuzzle, algorithmName,
heuristicType.toString());
    }
}
```

4.2.2. Searching

4.2.2.1. AStarSolver.java

```
package com.java.searching;

import com.java.checker.PuzzleChecker;
import com.java.exception.InvalidConfigurationException;
import java.util.*;

public class AStarSolver implements SearchStrategy {
```

```
    public SolverResult solve(State start) {
        long t0 = System.currentTimeMillis();
        PriorityQueue<Node> open = new
PriorityQueue<>(Comparator.comparingInt(n->n.f));
        Map<State, Integer> bestG = new HashMap<>();
        open.add(new Node(start, null, null, 0, start.heuristic()));
        bestG.put(start,0);
        long nodes = 0;
        while(!open.isEmpty()){
            Node cur = open.poll();
            boolean isGoal;
            try {
                isGoal = PuzzleChecker.checkSolved(cur.state.getBoard());
            } catch (InvalidConfigurationException e) {
                isGoal = false;
            }
            if (isGoal) {
                List<Move> path = new ArrayList<>();
                for(Node n=cur;n.move!=null;n=n.parent) path.add(n.move);
                Collections.reverse(path);
                return new SolverResult(path, nodes, System.currentTimeMillis()-t0);
            }
            nodes++;
            for(Move mv:cur.state.successors()){
                int g2 = cur.g+1;
                if(bestG.getOrDefault(mv.next, Integer.MAX_VALUE)>g2){
                    bestG.put(mv.next,g2);
                    open.add(new Node(mv.next, cur, mv, g2, mv.next.heuristic()));
                }
            }
        }
        return new SolverResult(List.of(), nodes, System.currentTimeMillis()-t0);
    }
}
```

4.2.2.2. ConfiguredClone.java

```
package com.java.searching;

import com.java.model.*;
import java.util.Map;

public class ConfiguredClone {
    public static Board cloneBoard(Board b) {
        int R=b.getRows(), C=b.getCols();
        Board copy=new Board(R,C);
        for(Map.Entry<Character,Piece> e:b.getPieces().entrySet()){
            Piece p=e.getValue();
            for(int[] cell:p.occupiedCells()){
                copy.parseCell(cell[0],cell[1],p.getId());
            }
        }
        copy.setExit(b.getExitRow(),b.getExitCol());
        return copy;
    }
```

```
}
```

4.2.2.3. GBFSolver.java

```
package com.java.searching;

import com.java.checker.PuzzleChecker;
import com.java.exception.InvalidConfigurationException;
import java.util.*;


public class GBFSolver implements SearchStrategy {
   public SolverResult solve(State start) {
      long t0=System.currentTimeMillis();
      PriorityQueue<Node> open = new
PriorityQueue<>(Comparator.comparingInt(n->n.h));
      Set<State> closed = new HashSet<>();
      open.add(new Node(start,null,null,0,start.heuristic()));
      long nodes=0;
      while(!open.isEmpty()){
         Node cur=open.poll();
         boolean isGoal;
         try {
            isGoal = PuzzleChecker.checkSolved(cur.state.getBoard());
         } catch (InvalidConfigurationException e) {
            isGoal = false;
         }
         if (isGoal) {
            List<Move> path=new ArrayList<>();
            for(Node n=cur;n.move!=null;n=n.parent)path.add(n.move);
            Collections.reverse(path);
            return new SolverResult(path,nodes,System.currentTimeMillis()-t0);
         }
         nodes++;
         closed.add(cur.state);
         List<Node> successors = new ArrayList<>();
         for(Move mv:cur.state.successors()){
            if(!closed.contains(mv.next)){
               successors.add(new Node(mv.next,cur,mv,0,mv.next.heuristic()));
            }
         }
         // Limit successors to the k most promising ones
         successors.sort(Comparator.comparingInt(n -> n.h));
         for (int i = 0; i < Math.min(3, successors.size()); i++) {
            open.add(successors.get(i));
         }
      }
      return new SolverResult(List.of(),nodes,System.currentTimeMillis()-t0);
   }
}
```

4.2.2.4. IDAStarSolver.java

```
package com.java.searching;
```

```
import com.java.checker.PuzzleChecker;
import com.java.exception.InvalidConfigurationException;
import java.util.*;

public class IDAStarSolver implements SearchStrategy {
    public SolverResult solve(State start) {
        long t0 = System.currentTimeMillis();
        int threshold = start.heuristic();
        long nodes = 0;

        List<Move> path = new ArrayList<>();

        int iteration = 0;
        int maxIterations = 100; // Safety limit for extremely difficult puzzles

        while (iteration++ < maxIterations) {
            path.clear();

            Map<String, Integer> bestCost = new HashMap<>();

            Result result = new Result();
            result.nodesExpanded = nodes;

            boolean found = dfs(start, 0, threshold, path, bestCost, result);
            nodes = result.nodesExpanded;

            if (found) {
                return new SolverResult(path, nodes, System.currentTimeMillis() - t0);
            }

            if (result.nextThreshold == Integer.MAX_VALUE) {
                return new SolverResult(new ArrayList<>(), nodes, System.currentTimeMillis() -
t0);
            }

            threshold = result.nextThreshold;

            System.out.println("IDA* iteration " + iteration + ": increased threshold to " +
threshold);
        }

        // If we hit iteration limit, return best effort
        return new SolverResult(new ArrayList<>(), nodes, System.currentTimeMillis() - t0);
    }

    private static class Result {
        int nextThreshold = Integer.MAX_VALUE;
        long nodesExpanded = 0;
    }

    private boolean dfs(State state, int g, int threshold, List<Move> path,
                        Map<String, Integer> bestCost, Result result) {
        result.nodesExpanded++;

        int h = state.heuristic();
        int f = g + h;
```

```
        // Prune if f-value exceeds threshold
        if (f > threshold) {
            result.nextThreshold = Math.min(result.nextThreshold, f);
            return false;
        }

        // Check for goal state
        boolean isGoal;
        try {
            isGoal = PuzzleChecker.checkSolved(state.getBoard());
        } catch (InvalidConfigurationException e) {
            isGoal = false;
        }

        if (isGoal) {
            return true;
        }

        String stateKey = state.toString();

        Integer previousCost = bestCost.get(stateKey);
        if (previousCost != null && previousCost <= g) {
            return false;
        }

        bestCost.put(stateKey, g);

        List<Move> successors = state.successors();
        successors.sort(Comparator.comparingInt(move -> g + 1 + move.next.heuristic()));

        // Try each successor
        for (Move move : successors) {
            path.add(move);

            if (dfs(move.next, g + 1, threshold, path, bestCost, result)) {
                return true;
            }

            path.remove(path.size() - 1);
        }

        return false;
    }
}
```

4.2.2.5. Move.java

```
package com.java.searching;

public class Move {
    public final int dr, dc;
    public final char id;
    public final State next;
    public Move(int dr,int dc,char id,State next){
```

```
        this.dr=dr;this.dc=dc;this.id=id;this.next=next;
    }
}
```

4.2.2.6. Node.java

```
package com.java.searching;

public class Node {
    public final State state;
    public final Node parent;
    public final Move move;
    public final int g, h, f;
    public Node(State s, Node p, Move m, int g, int h){
        this.state=s;this.parent=p;this.move=m;this.g=g;this.h=h;this.f=g+h;
    }
}
```

4.2.2.7. SearchStrategy.java

```
package com.java.searching;

public interface SearchStrategy {
    SolverResult solve(State start);
}
```

4.2.2.8. SolverResult.java

```
package com.java.searching;

import java.util.*;

public class SolverResult {
    public final List<Move> path;
    public final long nodesCreated;
    public final long timeMs;
    public SolverResult(List<Move> path, long nodesCreated, long timeMs){
        this.path=path;this.nodesCreated=nodesCreated;this.timeMs=timeMs;
    }
}
```

4.2.2.9. State.java

```
package com.java.searching;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.heuristic.*;
import java.util.*;

public class State {
    private final Board board;
    private final String key;
    private static HeuristicType currentHeuristic = HeuristicType.COMPOSITE;
```

```
    public State(Board b) {
        this.board = b;
        StringBuilder sb = new StringBuilder();
        for (int r = 0; r < b.getRows(); r++) {
            for (int c = 0; c < b.getCols(); c++) {
                sb.append(b.getCell(r, c));
            }
        }
        sb.append("K").append(b.getExitRow()).append(",").append(b.getExitCol());
        this.key = sb.toString();
    }
    public Board getBoard() { return board; }
    public List<Move> successors() {
        List<Move> moves = new ArrayList<>();
        for (Piece p : board.getPieces().values()) {
            for (int[] d : new int[][]{{0,1},{0,-1},{1,0},{-1,0}}) {
                try {
                    Board copy = ConfiguredClone.cloneBoard(board);
                    copy.movePiece(copy.getPieces().get(p.getId()), d[0], d[1]);
                    moves.add(new Move(d[0], d[1], p.getId(), new State(copy)));
                } catch (Exception ignored) {}
            }
        }
        return moves;
    }

    public static void setHeuristic(HeuristicType heuristic) {
        currentHeuristic = heuristic;
    }

    public int heuristic() {
        switch (currentHeuristic) {
            case PATTERN:
                return new Pattern().calculate(this);
            case MANHATTAN:
                return new Manhattan().calculate(this);
            case BLOCKING:
                return new EBlocking().calculate(this);
            default:
                return new Composite().calculate(this);

        }
    }

    @Override public boolean equals(Object o) {
        return o instanceof State && ((State)o).key.equals(key);
    }
    @Override public int hashCode() {
        return key.hashCode();
    }

    public static String getHeuristicType() {
        return currentHeuristic.toString();
    }
}
```

4.2.2.10. UCSolver.java

```
package com.java.searching;

import com.java.checker.PuzzleChecker;
import com.java.exception.InvalidConfigurationException;
import java.util.*;


public class UCSolver implements SearchStrategy {
   @Override
   public SolverResult solve(State start) {
      long t0=System.currentTimeMillis();
      PriorityQueue<Node> open = new
PriorityQueue<>(Comparator.comparingInt(n->n.g));
      Map<State,Integer> bestG=new HashMap<>();
      open.add(new Node(start,null,null,0,0));
      bestG.put(start,0);
      long nodes=0;
      while(!open.isEmpty()){
         Node cur=open.poll();
         boolean isGoal;
         try {
            isGoal = PuzzleChecker.checkSolved(cur.state.getBoard());
         } catch (InvalidConfigurationException e) {
            isGoal = false;
         }
         if (isGoal) {
            List<Move> path=new ArrayList<>();
            for(Node n=cur;n.move!=null;n=n.parent)path.add(n.move);
            Collections.reverse(path);
            return new SolverResult(path,nodes,System.currentTimeMillis()-t0);
         }
         nodes++;
         for(Move mv:cur.state.successors()){
            int g2=cur.g+1;
            if(bestG.getOrDefault(mv.next,Integer.MAX_VALUE)>g2){
               bestG.put(mv.next,g2);
               open.add(new Node(mv.next,cur,mv,g2,0));
            }
         }
      }
      return new SolverResult(List.of(),nodes,System.currentTimeMillis()-t0);
   }
}
```

4.2.3. Heuristic

4.2.3.1. Composite.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.HashSet;
```

```
import java.util.Set;

public class Composite implements Heuristic {
   public int calculate(State state) {
      Board board = state.getBoard();
      Piece p = board.getPieces().get('P');
      int er = board.getExitRow(), ec = board.getExitCol();
      int br = p.getRow(), bc = p.getCol(), sz = p.getSize();
      boolean horiz = p.isHorizontal();
      int frontR = horiz ? br : br + sz - 1;
      int frontC = horiz ? bc + sz - 1 : bc;
      int dist = Math.abs(horiz ? (ec - frontC) : (er - frontR));
      Set<Character> blockers = new HashSet<>();
      if (horiz) {
         int start = Math.min(frontC, ec), end = Math.max(frontC, ec);
         for (int c = start + 1; c < end; c++) {
            char ch = board.getCell(br, c);
            if (ch != '.') blockers.add(ch);
         }
      } else {
         int start = Math.min(frontR, er), end = Math.max(frontR, er);
         for (int r = start + 1; r < end; r++) {
            char ch = board.getCell(r, bc);
            if (ch != '.') blockers.add(ch);
         }
      }
      return dist + blockers.size();
   }
}
```

4.2.3.2. EBlocking.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.*;

public class EBlocking implements Heuristic {
   @Override
   public int calculate(State state) {
      Board board = state.getBoard();
      Piece p = board.getPieces().get('P');
      int er = board.getExitRow();
      int ec = board.getExitCol();

      int targetDistance = distanceToExit(board, p, er, ec);

      Set<Character> directBlockers = countDirectBlockers(board, p, er, ec);

      int recursiveBlockers = countRecursiveBlockers(board, directBlockers);

      int mobility = calculateMobilityScore(board, p, directBlockers);
```

```
    return (targetDistance * 4) +
           (directBlockers.size() * 3) +
           (recursiveBlockers * 2) -
           (int)(mobility * 0.5);
}

private int distanceToExit(Board board, Piece p, int er, int ec) {
    if (p.isHorizontal()) {
        int frontCol = p.getCol() + p.getSize() - 1;
        return Math.abs(ec - frontCol);
    } else {
        int frontRow = p.getRow() + p.getSize() - 1;
        return Math.abs(er - frontRow);
    }
}

private Set<Character> countDirectBlockers(Board board, Piece p, int er, int ec) {
    Set<Character> blockers = new HashSet<>();
    int rows = board.getRows();
    int cols = board.getCols();

    if (p.isHorizontal()) {
        int row = p.getRow();
        int frontCol = p.getCol() + p.getSize() - 1;

        if (ec > frontCol) {
            for (int c = frontCol + 1; c < cols; c++) {
                char cell = board.getCell(row, c);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
        } else if (ec < p.getCol()) {
            for (int c = Math.max(0, ec); c < p.getCol(); c++) {
                char cell = board.getCell(row, c);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
        }
    } else {
        int col = p.getCol();
        int frontRow = p.getRow() + p.getSize() - 1;

        if (er > frontRow) {
            for (int r = frontRow + 1; r < rows; r++) {
                char cell = board.getCell(r, col);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
        } else if (er < p.getRow()) {
            for (int r = Math.max(0, er); r < p.getRow(); r++) {
                char cell = board.getCell(r, col);
                if (cell != '.') {
                    blockers.add(cell);
```

```
            }
        }
    }
}

return blockers;
}

private int countRecursiveBlockers(Board board, Set<Character> directBlockers) {
    int recursiveCount = 0;
    Map<Character, Piece> pieces = board.getPieces();

    for (char blockerId : directBlockers) {
        Piece blocker = pieces.get(blockerId);
        if (blocker == null) continue;

        Set<Character> secondaryBlockers = findPiecesBlocking(board, blocker);
        recursiveCount += secondaryBlockers.size();
    }

    return recursiveCount;
}

private Set<Character> findPiecesBlocking(Board board, Piece piece) {
    Set<Character> blockingPieces = new HashSet<>();

    if (piece.isHorizontal()) {
        int row = piece.getRow();

        if (piece.getCol() > 0) {
            char leftCell = board.getCell(row, piece.getCol() - 1);
            if (leftCell != '.') {
                blockingPieces.add(leftCell);
            }
        }

        int rightPos = piece.getCol() + piece.getSize();
        if (rightPos < board.getCols()) {
            char rightCell = board.getCell(row, rightPos);
            if (rightCell != '.') {
                blockingPieces.add(rightCell);
            }
        }
    } else {
        int col = piece.getCol();

        if (piece.getRow() > 0) {
            char upCell = board.getCell(piece.getRow() - 1, col);
            if (upCell != '.') {
                blockingPieces.add(upCell);
            }
        }

        int bottomPos = piece.getRow() + piece.getSize();
        if (bottomPos < board.getRows()) {
            char downCell = board.getCell(bottomPos, col);
```

```
            if (downCell != '.') {
                blockingPieces.add(downCell);
            }
        }
    }

    return blockingPieces;
  }

  private int calculateMobilityScore(Board board, Piece p, Set<Character> directBlockers)
{
    int mobilityScore = 0;
    Map<Character, Piece> pieces = board.getPieces();

    int targetMobility = calculatePieceMobility(board, p);
    mobilityScore += targetMobility * 2;

    for (char blockerId : directBlockers) {
        Piece blocker = pieces.get(blockerId);
        if (blocker == null) continue;

        int blockerMobility = calculatePieceMobility(board, blocker);
        mobilityScore += blockerMobility;
    }

    return mobilityScore;
  }

  private int calculatePieceMobility(Board board, Piece piece) {
    int mobility = 0;

    if (piece.isHorizontal()) {
        int row = piece.getRow();

        for (int c = piece.getCol() - 1; c >= 0; c--) {
            if (board.getCell(row, c) == '.') {
                mobility++;
            } else {
                break;
            }
        }

        for (int c = piece.getCol() + piece.getSize(); c < board.getCols(); c++) {
            if (board.getCell(row, c) == '.') {
                mobility++;
            } else {
                break;
            }
        }
    } else {
        int col = piece.getCol();

        for (int r = piece.getRow() - 1; r >= 0; r--) {
            if (board.getCell(r, col) == '.') {
                mobility++;
            } else {
```

```
                break;
            }
        }

        for (int r = piece.getRow() + piece.getSize(); r < board.getRows(); r++) {
            if (board.getCell(r, col) == '.') {
                mobility++;
            } else {
                break;
            }
        }
    }

    return mobility;
  }
}
```

4.2.3.3. Heuristic.java

```
package com.java.searching.heuristic;

import com.java.searching.State;

public interface Heuristic {
  int calculate (State state);
}
```

4.2.3.4. HeuristicType.java

```
package com.java.searching.heuristic;

import com.java.searching.State;

public enum HeuristicType {
  COMPOSITE, // Original heuristic (Distance + Blockers)
  MANHATTAN, // Manhattan distance heuristic
  PATTERN, // Pattern database heuristic
  BLOCKING // Enhanced blocking heuristic
}
```

4.2.3.5. Manhattan.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.HashSet;
import java.util.Set;

public class Manhattan implements Heuristic {
  public int calculate(State state) {
    Board board = state.getBoard();
    Piece p = board.getPieces().get('P');
    int er = board.getExitRow(), ec = board.getExitCol();
```

```
        if (p.isHorizontal()) {
            int pmr = p.getRow();
            int pmc = p.getCol() + (p.getSize() - 1) / 2;

            return Math.abs(ec - pmc) + Math.abs(er - pmr);
        }
        else {
            int pmr = p.getRow() + (p.getSize() - 1) / 2;
            int pmc = p.getCol();

            return Math.abs(ec - pmc) + Math.abs(er - pmr);
        }
    }
}
```

4.2.3.6. Pattern.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.*;
import java.io.*;

public class Pattern implements Heuristic {
    private static Map<String, Integer> patternDatabase = new HashMap<>();

    private static List<Set<Character>> patterns = new ArrayList<>();

    static {
        initializePatterns();
        loadPatternDatabase();
    }

    public int calculate(State state) {
        int total = 0;
        Board board = state.getBoard();

        for (Set<Character> pattern : patterns) {
            String patternHash = extractPatternHash(board, pattern);

            if (patternDatabase.containsKey(patternHash)) {
                total += patternDatabase.get(patternHash);
            } else {
                total += calculateBaseHeuristic(board, pattern);
            }
        }
        return total;
    }

    private static void initializePatterns() {
        patterns.add(new HashSet<>(Arrays.asList('P')));
        Set<Character> exitRegionPattern = new HashSet<>();
```

```
        exitRegionPattern.add('P');
        patterns.add(exitRegionPattern);
        Set<Character> cornerPattern = new HashSet<>();
        patterns.add(cornerPattern);
    }

    private static String getDbFilePath() {
        String homeDir = System.getProperty("user.home");
        return homeDir + File.separator + "rushHourPatternDB.ser";
    }

    private static void loadPatternDatabase() {
        try {
            File dbFile = new File(getDbFilePath());
            if (dbFile.exists()) {
                ObjectInputStream ois = new ObjectInputStream(new FileInputStream(dbFile));
                patternDatabase = (Map<String, Integer>) ois.readObject();
                ois.close();
                System.out.println("Loaded pattern database with " + patternDatabase.size() + "
entries");
            } else {
                System.out.println("No pattern database found, using fallback heuristics");
            }
        } catch (Exception e) {
            System.out.println("Error loading pattern database: " + e.getMessage());
        }
    }

    private String extractPatternHash(Board board, Set<Character> pattern) {
        StringBuilder hash = new StringBuilder();

        Piece p = board.getPieces().get('P');
        int er = board.getExitRow();
        int ec = board.getExitCol();

        if (pattern.contains('P')) {
            hash.append("P:")
                .append(p.getRow()).append(",")
                .append(p.getCol()).append(",")
                .append(p.getSize()).append(",")
                .append(p.isHorizontal() ? "H" : "V").append(";");
        }

        hash.append("K:").append(er).append(",").append(ec).append(";");

        if (pattern == patterns.get(1)) {
            if (p.isHorizontal()) {
                int row = p.getRow();
                for (int c = 0; c < board.getCols(); c++) {
                    char cell = board.getCell(row, c);
                    if (cell != '.' && cell != 'P' && !pattern.contains(cell)) {
                        pattern.add(cell);
                    }
                }
            }
            else {
```

```
            int col = p.getCol();
            for (int r = 0; r < board.getRows(); r++) {
                char cell = board.getCell(r, col);
                if (cell != '.' && cell != 'P' && !pattern.contains(cell)) {
                    pattern.add(cell);
                }
            }
        }
    }

    if (pattern == patterns.get(2)) {
        if (board.getCell(0, 0) != '.') pattern.add(board.getCell(0, 0));
        if (board.getCell(0, board.getCols()-1) != '.') pattern.add(board.getCell(0,
board.getCols()-1));
        if (board.getCell(board.getRows()-1, 0) != '.')
pattern.add(board.getCell(board.getRows()-1, 0));
        if (board.getCell(board.getRows()-1, board.getCols()-1) != '.')
            pattern.add(board.getCell(board.getRows()-1, board.getCols()-1));
    }

    for (char pieceId : pattern) {
        if (pieceId == 'P') continue;

        Piece piece = board.getPieces().get(pieceId);
        if (piece != null) {
            hash.append(pieceId).append(":")
                .append(piece.getRow()).append(",")
                .append(piece.getCol()).append(",")
                .append(piece.getSize()).append(",")
                .append(piece.isHorizontal() ? "H" : "V").append(";");
        }
    }
    return hash.toString();
  }

  private int calculateBaseHeuristic(Board board, Set<Character> pattern) {
    Piece p = board.getPieces().get('P');
    int er = board.getExitRow();
    int ec = board.getExitCol();

    if (pattern == patterns.get(0)) {
        if (p.isHorizontal()) {
            int frontCol = p.getCol() + p.getSize() - 1;
            return Math.abs(ec - frontCol);
        } else {
            int frontRow = p.getRow() + p.getSize() - 1;
            return Math.abs(er - frontRow);
        }
    }
    else if (pattern == patterns.get(1)) {
        Set<Character> blockers = new HashSet<>();

        if (p.isHorizontal()) {
            int row = p.getRow();
            int frontCol = p.getCol() + p.getSize() - 1;
            int start = Math.min(frontCol, ec);
```

```
            int end = Math.max(frontCol, ec);

            for (int c = start + 1; c < end; c++) {
                char cell = board.getCell(row, c);
                if (cell != '.') blockers.add(cell);
            }
        } else {
            int col = p.getCol();
            int frontRow = p.getRow() + p.getSize() - 1;
            int start = Math.min(frontRow, er);
            int end = Math.max(frontRow, er);

            for (int r = start + 1; r < end; r++) {
                char cell = board.getCell(r, col);
                if (cell != '.') blockers.add(cell);
            }
        }

        return blockers.size() * 2;
    }
    else if (pattern == patterns.get(2)) {
        int penalty = 0;
        int rows = board.getRows();
        int cols = board.getCols();

        if (board.getCell(0, 0) != '.' && board.getCell(0, 1) != '.' &&
            board.getCell(1, 0) != '.') {
            penalty += 1;
        }

        if (board.getCell(0, cols-1) != '.' && board.getCell(0, cols-2) != '.' &&
            board.getCell(1, cols-1) != '.') {
            penalty += 1;
        }

        if (board.getCell(rows-1, 0) != '.' && board.getCell(rows-1, 1) != '.' &&
            board.getCell(rows-2, 0) != '.') {
            penalty += 1;
        }

        if (board.getCell(rows-1, cols-1) != '.' && board.getCell(rows-1, cols-2) != '.' &&
            board.getCell(rows-2, cols-1) != '.') {
            penalty += 1;
        }
        return penalty;
    }

    return 0;
  }

  public static void buildPatternDatabase() {
    System.out.println("Building pattern database - this may take a while...");

    try {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(getDbFilePath()));
```

```
            oos.writeObject(patternDatabase);
            oos.close();
            System.out.println("Saved pattern database with " + patternDatabase.size() + "
entries");
        } catch (IOException e) {
            System.out.println("Error saving pattern database: " + e.getMessage());
        }
    }

    private static class PatternState implements Serializable {
        Map<Character, Piece> pieces;
        int er, ec;

        public PatternState(Map<Character, Piece> pieces, int er, int ec) {
            this.pieces = pieces;
            this.er = er;
            this.ec = ec;
        }
    }
}
```

4.2.4. Checker

4.2.4.1. PuzzleChecker.java

```
package com.java.checker;

import com.java.exception.InvalidConfigurationException;
import com.java.model.Board;
import com.java.model.Piece;

public class PuzzleChecker {
    /**
     * Returns true if:
     *  - P actually covers the exit (rare, since exit is outside), or
     *  - there's a straight, empty corridor from P to K in P's orientation.
     *
     * Throws if the primary piece is oriented wrong for that exit.
     */
    public static boolean checkSolved(Board board)
      throws InvalidConfigurationException
    {
        Piece p = board.getPieces().get('P');
        if (p == null) {
            throw new InvalidConfigurationException("No primary piece 'P' found");
        }

        int er = board.getExitRow(), ec = board.getExitCol();
        int R  = board.getRows(), C  = board.getCols();

        if (er == Integer.MIN_VALUE || ec == Integer.MIN_VALUE) {
            throw new InvalidConfigurationException("No exit 'K' defined");
        }

        boolean exitLeft   = ec < 0;
```

```
        boolean exitRight  = ec > C - 1;
        boolean exitTop    = er < 0;
        boolean exitBottom = er > R - 1;

        if (( (exitLeft || exitRight) && p.isVertical() ) ||
            ( (exitTop  || exitBottom) && p.isHorizontal() ))
        {
            throw new InvalidConfigurationException(
                "Primary Vehicles Cannot Exit the Compound"
            );
        }

        for (int[] cell : p.occupiedCells()) {
            if (cell[0] == er && cell[1] == ec) {
                System.out.println("Puzzle solved!");
                return true;
            }
        }

        if ((exitLeft || exitRight) && p.isHorizontal() && p.getRow() == er) {
            int start = exitLeft ? 0 : p.getCol() + p.getSize();
            int end   = exitLeft ? p.getCol() : C;
            for (int c = start; c < end; c++) {
                if (board.getCell(er, c) != '.') {
                    return false;
                }
            }
            System.out.println("Puzzle solved!");
            return true;
        }

        if ((exitTop || exitBottom) && p.isVertical() && p.getCol() == ec) {
            int start = exitTop    ? 0 : p.getRow() + p.getSize();
            int end   = exitTop    ? p.getRow() : R;
            for (int r = start; r < end; r++) {
                if (board.getCell(r, ec) != '.') {
                    return false;
                }
            }
            System.out.println("Puzzle solved!");
            return true;
        }

        return false;
    }
}
```

4.2.5. Exception

4.2.5.1. InvalidConfigurationException.java

```
package com.java.exception;

public class InvalidConfigurationException extends Exception {
    public InvalidConfigurationException(String message) {
```

```
        super(message);
    };
}
```

4.2.5.2. MoveBlockedException.java

```
package com.java.exception;

import com.java.model.Piece;

public class MoveBlockedException extends Exception {
    private final Piece blocker;

    public MoveBlockedException(Piece blocker) {
        super("Move blocked by piece '" + (blocker != null ? blocker.getId() : "unknown") + "'");
        this.blocker = blocker;
    }

    /** The piece that's blocking. */
    public Piece getBlocker() {
        return blocker;
    }
}
```

4.2.5.3. PuzzleNotSolvedException.java

```
package com.java.exception;

public class PuzzleNotSolvedException extends Exception {
    public PuzzleNotSolvedException() {
        super("Puzzle not solved");
    }
}
```

4.2.6. GUI

4.2.6.1. BoardPanel.java

```
package com.java.gui;

import com.java.model.Board;
import javax.swing.*;
import java.awt.*;
import java.util.HashMap;
import java.util.Map;

public class BoardPanel extends JPanel {
    private Board board;
    private Map<Character, Color> pieceColors;

    public BoardPanel() {
        setPreferredSize(new Dimension(400, 400));
        setBackground(Color.WHITE);
        initializeColors();
    }
```

```
private void initializeColors() {
    pieceColors = new HashMap<>();

    // Primary piece is always red
    pieceColors.put('P', Color.RED);

    // Basic colors
    pieceColors.put('A', Color.GREEN);
    pieceColors.put('B', Color.BLUE);
    pieceColors.put('C', Color.YELLOW);
    pieceColors.put('D', Color.CYAN);
    pieceColors.put('E', Color.MAGENTA);
    pieceColors.put('F', Color.ORANGE);
    pieceColors.put('G', Color.PINK);
    pieceColors.put('H', Color.LIGHT_GRAY);

    // Extended colors - each with a unique RGB value
    pieceColors.put('I', new Color(139, 69, 19));    // Brown
    pieceColors.put('J', new Color(0, 100, 0));      // Dark Green
    pieceColors.put('K', new Color(70, 130, 180));   // Steel Blue
    pieceColors.put('L', new Color(148, 0, 211));    // Dark Violet
    pieceColors.put('M', new Color(255, 140, 0));    // Dark Orange
    pieceColors.put('N', new Color(255, 20, 147));   // Deep Pink
    pieceColors.put('O', new Color(72, 61, 139));    // Dark Slate Blue
    pieceColors.put('Q', new Color(255, 99, 71));    // Tomato
    pieceColors.put('R', new Color(30, 144, 255));   // Dodger Blue
    pieceColors.put('S', new Color(154, 205, 50));   // Yellow Green
    pieceColors.put('T', new Color(255, 218, 185));  // Peach
    pieceColors.put('U', new Color(221, 160, 221));  // Plum
    pieceColors.put('V', new Color(176, 224, 230));  // Powder Blue
    pieceColors.put('W', new Color(210, 180, 140));  // Tan
    pieceColors.put('Y', new Color(255, 182, 193));  // Light Pink
    pieceColors.put('Z', new Color(135, 206, 235));  // Sky Blue

    // Exit marker
    pieceColors.put('K', Color.GRAY);
}

public void setBoard(Board board) {
    this.board = board;
    repaint();
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    if (board == null) {
        g.setColor(Color.GRAY);
        g.setFont(new Font("Arial", Font.BOLD, 16));
        g.drawString("Please load a puzzle file", 50, 50);
        return;
    }

    int rows = board.getRows();
```

```
int cols = board.getCols();

int cellSize = Math.min(getWidth() / cols, getHeight() / rows);
int boardWidth = cellSize * cols;
int boardHeight = cellSize * rows;

int startK = (getWidth() - boardWidth) / 2;
int startY = (getHeight() - boardHeight) / 2;

g.setColor(Color.BLACK);
for (int r = 0; r <= rows; r++) {
   g.drawLine(startK, startY + r * cellSize,
           startK + cols * cellSize, startY + r * cellSize);
}
for (int c = 0; c <= cols; c++) {
   g.drawLine(startK + c * cellSize, startY,
           startK + c * cellSize, startY + rows * cellSize);
}

int exitRow = board.getExitRow();
int exitCol = board.getExitCol();

g.setColor(Color.RED);
if (exitCol == cols) { // Exit on right
   g.fillRect(startK + cols * cellSize, startY + exitRow * cellSize,
           cellSize/2, cellSize);
} else if (exitCol == -1) { // Exit on left
   g.fillRect(startK - cellSize/2, startY + exitRow * cellSize,
           cellSize/2, cellSize);
} else if (exitRow == rows) { // Exit on bottom
   g.fillRect(startK + exitCol * cellSize, startY + rows * cellSize,
           cellSize, cellSize/2);
} else if (exitRow == -1) { // Exit on top
   g.fillRect(startK + exitCol * cellSize, startY - cellSize/2,
           cellSize, cellSize/2);
}

g.setColor(Color.WHITE);
g.drawString("EXIT", startK + exitCol * cellSize + cellSize/2 - 15,
        startY + exitRow * cellSize + cellSize/2 + 5);

for (int r = 0; r < rows; r++) {
   for (int c = 0; c < cols; c++) {
      char cell = board.getCell(r, c);
      if (cell != '.') {
         Color pieceColor = pieceColors.getOrDefault(cell, Color.GRAY);
         g.setColor(pieceColor);
         g.fillRect(startK + c * cellSize + 2,
                 startY + r * cellSize + 2,
                 cellSize - 4, cellSize - 4);
         g.setColor(Color.BLACK);
         g.drawString(String.valueOf(cell),
                  startK + c * cellSize + cellSize/2 - 5,
                  startY + r * cellSize + cellSize/2 + 5);
      }
   }
```

```
    }
  }
}
```

4.2.6.2. CustomFonts.java

```
package com.java.gui;

import java.awt.*;
import java.io.File;

public class CustomFonts {
  public static Font load(String path, float size, int style) {
    try {
      Font f = Font.createFont(Font.TRUETYPE_FONT, new File(path));
      return f.deriveFont(style, size);
    } catch (Exception e) {
      return new Font("Arial", style, (int)size);
    }
  }
}
```

4.2.6.3. InfoPanel.java

```
package com.java.gui;

import javax.swing.*;
import java.awt.*;

public class InfoPanel extends JPanel {
  private JLabel statusLabel;
  private JLabel timeLabel;
  private JLabel nodesLabel;
  private JLabel stepsLabel;

  public InfoPanel() {
    setPreferredSize(new Dimension(150, 0));
    setLayout(new GridLayout(4, 1));
    initializeComponents();
  }

  private void initializeComponents() {
    statusLabel = new JLabel("Status: Ready");
    timeLabel = new JLabel("Time: -");
    nodesLabel = new JLabel("Nodes: -");
    stepsLabel = new JLabel("Steps: -");

    statusLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    timeLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    nodesLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    stepsLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

    add(statusLabel);
    add(timeLabel);
    add(nodesLabel);
```

```
        add(stepsLabel);
    }

    public void updateStatus(String status) {
        statusLabel.setText("Status: " + status);
    }

    public void updateMetrics(long time, long nodes, int steps) {
        timeLabel.setText("Time: " + time + " ms");
        nodesLabel.setText("Nodes: " + nodes);
        stepsLabel.setText("Steps: " + steps);
    }

    public void clearMetrics() {
        timeLabel.setText("Time: -");
        nodesLabel.setText("Nodes: -");
        stepsLabel.setText("Steps: -");
    }
}
```

4.2.6.4. LoadingPanel.java

```
package com.java.gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import com.java.gui.CustomFonts;

public class LoadingPanel extends JPanel {
    public static final String LOADING = "LOADING";
    private JLabel label;
    private int dotCount = 0;

    public LoadingPanel() {
        setLayout(new GridBagLayout());
        setOpaque(false);

        Font font = CustomFonts.load("resources/fonts/MyFont.ttf", 32f, Font.PLAIN);
        label = new JLabel("Calculating");
        label.setFont(font);
        add(label);

        Timer timer = new Timer(500, e -> {
            dotCount = (dotCount + 1) % 4;
            label.setText("Calculating" + ".".repeat(dotCount));
        });
        timer.start();
    }
}
```

4.2.6.5. MainMenuPanel.java

```
package com.java.gui;
```

```
import com.java.model.Board;
import com.java.model.Config;
import com.java.searching.heuristic.HeuristicType;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.io.File;
import javafx.embed.swing.JFXPanel;

public class MainMenuPanel extends JPanel {
    public static final String MAIN = "MAIN_MENU";

    private RushHourGUI parent;
    private JButton uploadBtn, calculateBtn;
    private BoardPanel preview;
    private File currentFile;
    private Board loadedBoard;
    private JPanel contentPanel;

    public MainMenuPanel(RushHourGUI gui) {
        this.parent = gui;
        setLayout(new BorderLayout());
        setOpaque(false);

        // Create content panel first, before adding video background
        contentPanel = new JPanel(new BorderLayout());
        contentPanel.setOpaque(false);  // This is crucial!

        // Add top bar with Upload & Calculate buttons - ADD MORE PADDING
        JPanel top = new JPanel();
        top.setOpaque(false);  // Make transparent
        top.setBorder(BorderFactory.createEmptyBorder(135, 0, 20, 0)); // Add padding at top

        uploadBtn = new JButton("Upload a .txt File...");
        uploadBtn.addActionListener(e -> onUpload());
        top.add(uploadBtn);

        calculateBtn = new JButton("Calculate ▸");
        calculateBtn.setEnabled(false);
        calculateBtn.addActionListener(e -> onCalculate());
        top.add(calculateBtn);

        contentPanel.add(top, BorderLayout.NORTH);

        // Center preview - also make transparent with more spacing
        preview = new BoardPanel();
        JPanel center = new JPanel(new GridBagLayout());
        center.setOpaque(false);
        // Wrap the preview in another panel with some padding
        JPanel previewWrapper = new JPanel();
        previewWrapper.setOpaque(false);
        previewWrapper.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
        previewWrapper.add(preview);
        center.add(previewWrapper);
        contentPanel.add(center, BorderLayout.CENTER);
```

```
        // Try creating video background last (IMPORTANT)
        try {
            // Create a separate panel for the video
            JPanel videoPanel = new JPanel(new BorderLayout());
            videoPanel.setOpaque(false);
            JFXPanel fx = VideoBackground.create("resources/video/bg.mp4");
            videoPanel.add(fx, BorderLayout.CENTER);

            // Add video panel FIRST, then the content panel on top
            setLayout(new OverlayLayout(this)); // Use OverlayLayout for proper stacking
            add(contentPanel);  // Add content last so it's on top
            add(videoPanel);    // Add video first so it's on bottom
        } catch (Exception e) {
            e.printStackTrace();
            setBackground(Color.WHITE);
            add(contentPanel, BorderLayout.CENTER);  // Just add content without video
        }
    }

    private void onUpload() {
        JFileChooser chooser = new JFileChooser("test");
        chooser.setFileFilter(new FileNameExtensionFilter("Text Files", "txt"));
        if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            currentFile = chooser.getSelectedFile();
            try {
                loadedBoard = Config.loadConfig(currentFile.getPath());
                preview.setBoard(loadedBoard);
                calculateBtn.setEnabled(true);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Load error: " + ex.getMessage(),
                                              "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private void onCalculate() {
        String[] options = {"A★ Search", "Uniform Cost", "Greedy Best-First", "IDA★ Search"};
        String algo = (String) JOptionPane.showInputDialog(
            this,
            "Choose algorithm:",
            "Algorithm Selection",
            JOptionPane.PLAIN_MESSAGE,
            null,
            options,
            options[0]
        );

        if (algo != null) {
            // If not using UCS, ask for heuristic type
            HeuristicType heuristicType = HeuristicType.COMPOSITE; // Default

            if (!algo.equals("Uniform Cost")) {
                String[] heuristics = {"Composite", "Manhattan Distance", "Pattern Database",
"Enhanced Blocking"};
                String heuristic = (String) JOptionPane.showInputDialog(
                    this,
```

```
                "Choose heuristic:",
                "Heuristic Selection",
                JOptionPane.PLAIN_MESSAGE,
                null,
                heuristics,
                heuristics[0]
            );

            if (heuristic != null) {
                // Set heuristic based on selection
                switch (heuristic) {
                    case "Manhattan Distance":
                        heuristicType = HeuristicType.MANHATTAN;
                        break;
                    case "Pattern Database":
                        heuristicType = HeuristicType.PATTERN;
                        break;
                    case "Enhanced Blocking":
                        heuristicType = HeuristicType.BLOCKING;
                        break;
                    default:
                        heuristicType = HeuristicType.COMPOSITE;
                        break;
                }
            } else {
                return; // User canceled heuristic selection
            }
        }

        parent.showLoading();
        new SolverWorker(parent, loadedBoard, algo, heuristicType).execute();
    }
  }

  // Add this method to support loading a puzzle from a file
  public void loadPuzzleFile(File file) {
    currentFile = file;
    try {
        loadedBoard = Config.loadConfig(currentFile.getPath());
        preview.setBoard(loadedBoard);
        calculateBtn.setEnabled(true);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Load error: " + ex.getMessage(),
                                      "Error", JOptionPane.ERROR_MESSAGE);
    }
  }

  // Add this method to access the loaded board
  public Board getLoadedBoard() {
    return loadedBoard;
  }
}
```

4.2.6.6. RushHourGUI.java

```
package com.java.gui;

import java.awt.*;
import javax.swing.*;

public class RushHourGUI extends JFrame {
   private JPanel cards;
   private MainMenuPanel mainMenu;
   private LoadingPanel  loading;
   private SolvedPanel   solved;

   public RushHourGUI() {
      super("YuukaFinder: Rush Hour Solver");
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setSize(800, 600);
      setLocationRelativeTo(null);

      cards    = new JPanel(new CardLayout());
      mainMenu = new MainMenuPanel(this);
      loading  = new LoadingPanel();
      solved   = new SolvedPanel(this);

      cards.add(mainMenu, MainMenuPanel.MAIN);
      cards.add(loading,  LoadingPanel.LOADING);
      cards.add(solved,   SolvedPanel.SOLVED);

      setContentPane(cards);
   }

   public void showMain() {
      ((CardLayout)cards.getLayout()).show(cards, MainMenuPanel.MAIN);
   }

   public void showLoading() {
      ((CardLayout)cards.getLayout()).show(cards, LoadingPanel.LOADING);
   }

   public void showSolved() {
      ((CardLayout)cards.getLayout()).show(cards, SolvedPanel.SOLVED);
   }
}
```

4.2.6.7. SolvedPanel.java

```
package com.java.gui;

import com.java.searching.SolverResult;
import com.java.searching.Move;
import com.java.searching.State;
import com.java.searching.heuristic.HeuristicType;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
```

```
import java.io.File;
import java.util.List;
import javax.imageio.ImageIO;
import javafx.embed.swing.JFXPanel;

public class SolvedPanel extends JPanel {
    public static final String SOLVED = "SOLVED";

    private RushHourGUI parent;
    private BoardPanel boardView;
    private JLabel timeLbl, nodesLbl, stepsLbl;
    private JButton replayBtn, saveImgBtn, backBtn;
    private SolverResult result;
    private State initial;

    private String algorithmName = "Custom";
    private HeuristicType heuristicType = HeuristicType.COMPOSITE;

    public SolvedPanel(RushHourGUI gui) {
        this.parent = gui;
        setLayout(new BorderLayout());
        setOpaque(false);

        // video background
        try {
            JFXPanel fx = VideoBackground.create("resources/video/bg.mp4");
            add(fx, BorderLayout.CENTER);
        } catch (Exception e) {
            setBackground(Color.WHITE);
        }

        boardView = new BoardPanel();
        add(boardView, BorderLayout.CENTER);

        JPanel right = new JPanel(new GridLayout(7, 1, 5, 5));
        right.setOpaque(false);
        timeLbl  = new JLabel("Time: –");
        nodesLbl = new JLabel("Nodes: –");
        stepsLbl = new JLabel("Steps: –");
        right.add(timeLbl);
        right.add(nodesLbl);
        right.add(stepsLbl);

        replayBtn  = new JButton("Replay Solution");
        saveImgBtn = new JButton("Save Image");
        backBtn    = new JButton("Back");
        JButton saveBtn = new JButton("Save Solution");

        replayBtn .addActionListener(e -> replay());
        saveImgBtn.addActionListener(e -> saveImage());
        backBtn   .addActionListener(e -> parent.showMain());
        saveBtn.addActionListener(e -> saveSolution());

        right.add(replayBtn);
        right.add(saveImgBtn);
        right.add(backBtn);
```

```
        right.add(saveBtn);

        add(right, BorderLayout.EAST);
    }

    /** Called by SolverWorker when done. */
    public void setResult(State start, SolverResult res) {
        this.initial = start;
        this.result  = res;
        timeLbl .setText("Time: "  + res.timeMs    + " ms");
        nodesLbl.setText("Nodes: " + res.nodesCreated);
        stepsLbl.setText("Steps: " + res.path.size());

        boardView.setBoard(start.getBoard());
        parent.showSolved();
        replay();
    }

    private void replay() {
        List<Move> path = result.path;
        Timer timer = new Timer(500, null);
        final int[] idx = {0};
        timer.addActionListener(e -> {
            if (idx[0] >= path.size()) {
                timer.stop();
                return;
            }
            boardView.setBoard(path.get(idx[0]++).next.getBoard());
        });
        timer.start();
    }

    private void saveImage() {
        // Create solutions directory if it doesn't exist
        File solutionsDir = new File("test/solutions");
        if (!solutionsDir.exists()) {
            solutionsDir.mkdirs();
        }

        String filename = JOptionPane.showInputDialog(this,
            "Enter filename (without .png extension):",
            "solution");

        if (filename != null && !filename.trim().isEmpty()) {
            try {
                // Create screenshot
                BufferedImage img = new BufferedImage(getWidth(), getHeight(),
BufferedImage.TYPE_INT_ARGB);
                paint(img.getGraphics());

                // Save to the solutions directory with timestamp
                String timestamp = new
java.text.SimpleDateFormat("yyyyMMdd_HHmmss").format(new java.util.Date());
                String filepath = "test/solutions/" + filename + "_" + timestamp + ".png";
                ImageIO.write(img, "PNG", new File(filepath));
```

```
            JOptionPane.showMessageDialog(this,
                "Screenshot saved to: " + filepath,
                "Save Success", JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this,
                "Save failed: " + ex.getMessage(),
                "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
  }

  public void setAlgorithmInfo(String algo, HeuristicType heuristic) {
    this.algorithmName = algo;
    this.heuristicType = heuristic;
  }

  public void saveSolution() {
    if (result == null || initial == null) {
      JOptionPane.showMessageDialog(this, "No solution to save",
                                    "Error", JOptionPane.ERROR_MESSAGE);
      return;
    }

    String filename = JOptionPane.showInputDialog(this,
      "Enter filename (without .txt extension):",
      "solution");

    if (filename != null && !filename.trim().isEmpty()) {
      try {
        // Use String version to avoid ambiguity
        com.java.model.Save.saveSolution(
          result, initial, filename, "GUI",
          algorithmName, heuristicType.toString()
        );
        JOptionPane.showMessageDialog(this,
          "Solution saved to test/solutions/" + filename + ".txt",
          "Save Success", JOptionPane.INFORMATION_MESSAGE);
      } catch (Exception e) {
        JOptionPane.showMessageDialog(this,
          "Error saving solution: " + e.getMessage(),
          "Save Error", JOptionPane.ERROR_MESSAGE);
      }
    }
  }
}
```

4.2.6.8. SolverWorker.java

```
package com.java.gui;

import javax.swing.SwingWorker;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import com.java.searching.*;
```

```
import com.java.searching.heuristic.HeuristicType;
import com.java.model.Board;
import com.java.checker.PuzzleChecker;
import com.java.exception.InvalidConfigurationException;

public class SolverWorker extends SwingWorker<SolverResult, Void> {
    private final RushHourGUI gui;
    private final com.java.searching.State start; // Use fully qualified name
    private final String algoName;
    private final HeuristicType heuristicType;

    public SolverWorker(RushHourGUI gui, Board board, String algoName, HeuristicType
heuristicType) {
        this.gui = gui;
        this.start = new com.java.searching.State(board); // Fully qualified
        this.algoName = algoName;
        this.heuristicType = heuristicType;

        // Set the heuristic type (if not UCS)
        if (!algoName.startsWith("Uniform")) {
            com.java.searching.State.setHeuristic(heuristicType); // Fully qualified
        }
    }

    @Override
    protected SolverResult doInBackground() throws Exception {
        SearchStrategy solver;

        if (algoName.startsWith("A★") || algoName.startsWith("A*")) {
            solver = new AStarSolver();
        } else if (algoName.startsWith("Uniform")) {
            solver = new UCSolver();
        } else if (algoName.startsWith("IDA*")) {
            solver = new IDAStarSolver();
        } else {
            solver = new GBFSolver();
        }

        return solver.solve(start);
    }

    @Override
    protected void done() {
        try {
            SolverResult res = get();

            // Check if path is empty but determine if that's because:
            // 1. Already solved (should show solution screen)
            // 2. Unsolvable (should show error)
            if (res.path.isEmpty()) {
                try {
                    // Check if initial state is already solved
                    if (PuzzleChecker.checkSolved(start.getBoard())) {
                        // Already solved - proceed to solution screen
                        System.out.println("Puzzle was already in solved state!");
```

```
                    // Continue to solution screen
                    java.lang.reflect.Field cardsField =
RushHourGUI.class.getDeclaredField("cards");
                    cardsField.setAccessible(true);
                    JPanel cards = (JPanel) cardsField.get(gui);

                    SolvedPanel solvedPanel = (SolvedPanel) cards.getComponent(2);
                    solvedPanel.setResult(start, res);
                    solvedPanel.setAlgorithmInfo(algoName, heuristicType);
                    gui.showSolved();

                    // Add a special message
                    JOptionPane.showMessageDialog(gui,
                        "The puzzle was already solved! No moves needed.",
                        "Already Solved", JOptionPane.INFORMATION_MESSAGE);
                } else {
                    // Not already solved, but no path found = unsolvable
                    JOptionPane.showMessageDialog(gui,
                        "No solution found after exploring " + res.nodesCreated + " nodes.",
                        "Unsolvable Puzzle", JOptionPane.WARNING_MESSAGE);
                    gui.showMain();
                }
            } catch (InvalidConfigurationException e) {
                JOptionPane.showMessageDialog(gui,
                    "Invalid puzzle configuration: " + e.getMessage(),
                    "Configuration Error", JOptionPane.ERROR_MESSAGE);
                gui.showMain();
            }
        } else {
            // Normal case - solution found with moves
            java.lang.reflect.Field cardsField = RushHourGUI.class.getDeclaredField("cards");
            cardsField.setAccessible(true);
            JPanel cards = (JPanel) cardsField.get(gui);

            SolvedPanel solvedPanel = (SolvedPanel) cards.getComponent(2);
            solvedPanel.setResult(start, res);
            solvedPanel.setAlgorithmInfo(algoName, heuristicType);
            gui.showSolved();
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(gui, "Error: " + ex.getMessage(),
                                      "Solver Error", JOptionPane.ERROR_MESSAGE);
        ex.printStackTrace();
        gui.showMain();
    }
  }
}
```

4.2.6.9. VideoBackground.java

```
package com.java.gui;

import javafx.application.Platform;
import javafx.embed.swing.JFXPanel;
import javafx.scene.Scene;
```

```
import javafx.scene.layout.StackPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.util.Duration;

import java.io.File;

public class VideoBackground {
    private static MediaPlayer activePlayer;

    public static JFXPanel create(String filepath) {
        JFXPanel fxPanel = new JFXPanel();
        Platform.runLater(() -> {
            try {
                File file = new File(filepath);
                if (!file.exists()) {
                    System.err.println("Video file not found: " + file.getAbsolutePath());
                    return;
                }

                Media media = new Media(file.toURI().toString());
                activePlayer = new MediaPlayer(media);
                activePlayer.setCycleCount(MediaPlayer.INDEFINITE);
                activePlayer.setMute(true); // Mute the player immediately

                MediaView view = new MediaView(activePlayer);

                // Don't preserve ratio so it fills the panel
                view.setPreserveRatio(false);

                StackPane root = new StackPane(view);
                Scene scene = new Scene(root);

                // Make background transparent so it doesn't block other components
                root.setStyle("-fx-background-color: transparent;");
                scene.setFill(javafx.scene.paint.Color.TRANSPARENT);

                // Bind the MediaView size to the panel size
                view.fitWidthProperty().bind(root.widthProperty());
                view.fitHeightProperty().bind(root.heightProperty());

                fxPanel.setScene(scene);
                activePlayer.play();

                // Double-check muting after player starts
                activePlayer.setOnPlaying(() -> {
                    activePlayer.setMute(true);
                });
            } catch (Exception e) {
                System.err.println("Error loading video: " + e.getMessage());
                e.printStackTrace();
            }
        });
        return fxPanel;
    }
```

```
}
```

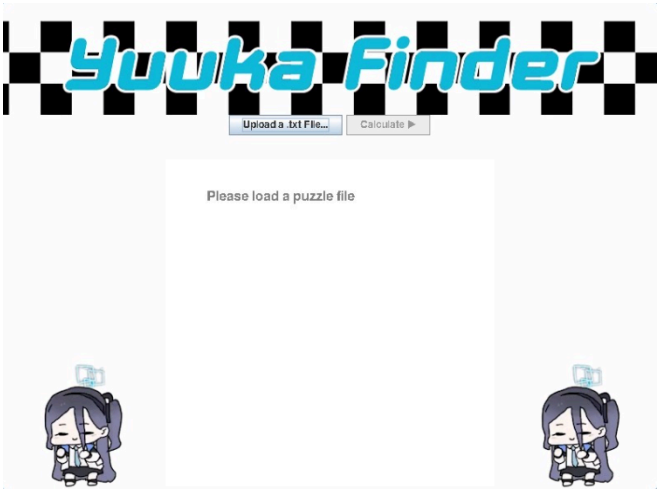
4.2.7. Main.java

```
package com.java;

import javax.swing.SwingUtilities;
import com.java.gui.RushHourGUI;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new RushHourGUI().setVisible(true);
        });
    }
}
```

BAB V
IMPLEMENTASI BONUS

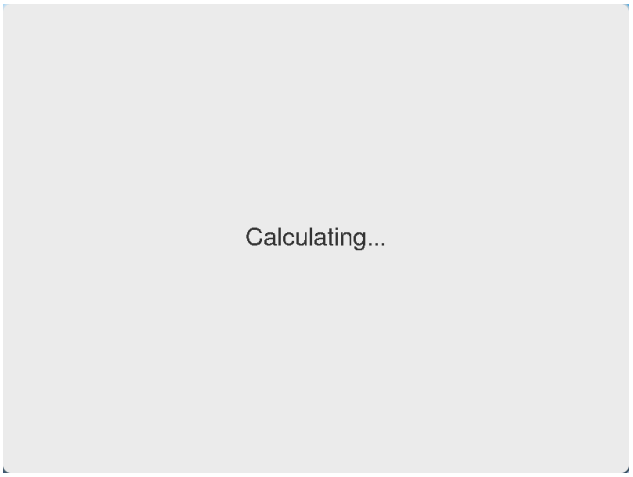
5.1. *Graphical User Interface (GUI)*

Dalam tugas kecil ini, kami berhasil mengimplementasikan bonus Graphical User Interface (GUI). Bonus ini dikerjakan dengan library JavaFX. GUI ini memiliki tiga window, yaitu menu utama, tampilan pencarian solusi (loading page) dan solusi.



Gambar 5.1.1. Main Menu GUI

Pada menu utama, pengguna dapat memilih konfigurasi puzzle dari file .txt. Konfigurasi yang tidak valid akan memunculkan pesan error. Pengguna juga dapat memilih algoritma yang akan digunakan dan heuristiknya (kalau heuristiknya memadai dengan algoritma yang dipilih). Setelah memilih algoritma dan puzzle, pengguna dapat memulai pencarian solusi.



Gambar 5.1.2. Loading GUI

Pencarian solusi akan dilakukan sementara GUI menampilkan pesan “Loading…” pada window tampilan pencarian solusi. Setelah selesai, GUI akan menampilkan window presentasi solusi.

Gambar 5.1.2. Result GUI

Hasil pencarian akan dipresentasikan dengan animasi sekuens status awal hingga status akhir. Animasi tersebut dapat diputar ulang menggunakan tombol ‘Replay Solution’. Presentasi hasil pencarian juga memberikan informasi berupa waktu pencarian, jumlah simpul, dan jumlah langkah. Untuk kembali ke menu utama, tombol ‘Back’ dapat ditekan.

5.2. Heuristik Alternatif

Dalam tugas kecil ini, kami berhasil mengimplementasikan bonus heuristik alternatif dengan mengimplementasikan tiga model heuristik yang dapat dipilih ketika menggunakan program, yaitu jumlah mobil yang menghalangi goal (EBlocking), Manhattan, dan pattern-recognition.

5.2.1. EBlocking.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.*;

public class EBlocking implements Heuristic {
    @Override
    public int calculate(State state) {
        Board board = state.getBoard();
        Piece p = board.getPieces().get('P');
        int er = board.getExitRow();
        int ec = board.getExitCol();

        int targetDistance = distanceToExit(board, p, er, ec);

        Set<Character> directBlockers = countDirectBlockers(board, p, er, ec);
```

```
    int recursiveBlockers = countRecursiveBlockers(board, directBlockers);

    int mobility = calculateMobilityScore(board, p, directBlockers);

    return (targetDistance * 4) +
           (directBlockers.size() * 3) +
           (recursiveBlockers * 2) -
           (int)(mobility * 0.5);
}

private int distanceToExit(Board board, Piece p, int er, int ec) {
    if (p.isHorizontal()) {
        int frontCol = p.getCol() + p.getSize() - 1;
        return Math.abs(ec - frontCol);
    } else {
        int frontRow = p.getRow() + p.getSize() - 1;
        return Math.abs(er - frontRow);
    }
}

private Set<Character> countDirectBlockers(Board board, Piece p, int er, int ec) {
    Set<Character> blockers = new HashSet<>();
    int rows = board.getRows();
    int cols = board.getCols();

    if (p.isHorizontal()) {
        int row = p.getRow();
        int frontCol = p.getCol() + p.getSize() - 1;

        if (ec > frontCol) {
            for (int c = frontCol + 1; c < cols; c++) {
                char cell = board.getCell(row, c);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
        } else if (ec < p.getCol()) {
            for (int c = Math.max(0, ec); c < p.getCol(); c++) {
                char cell = board.getCell(row, c);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
        }
    } else {
        int col = p.getCol();
        int frontRow = p.getRow() + p.getSize() - 1;

        if (er > frontRow) {
            for (int r = frontRow + 1; r < rows; r++) {
                char cell = board.getCell(r, col);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
```

```
        } else if (er < p.getRow()) {
            for (int r = Math.max(0, er); r < p.getRow(); r++) {
                char cell = board.getCell(r, col);
                if (cell != '.') {
                    blockers.add(cell);
                }
            }
        }
    }

    return blockers;
}

private int countRecursiveBlockers(Board board, Set<Character> directBlockers) {
    int recursiveCount = 0;
    Map<Character, Piece> pieces = board.getPieces();

    for (char blockerId : directBlockers) {
        Piece blocker = pieces.get(blockerId);
        if (blocker == null) continue;

        Set<Character> secondaryBlockers = findPiecesBlocking(board, blocker);
        recursiveCount += secondaryBlockers.size();
    }

    return recursiveCount;
}

private Set<Character> findPiecesBlocking(Board board, Piece piece) {
    Set<Character> blockingPieces = new HashSet<>();

    if (piece.isHorizontal()) {
        int row = piece.getRow();

        if (piece.getCol() > 0) {
            char leftCell = board.getCell(row, piece.getCol() - 1);
            if (leftCell != '.') {
                blockingPieces.add(leftCell);
            }
        }

        int rightPos = piece.getCol() + piece.getSize();
        if (rightPos < board.getCols()) {
            char rightCell = board.getCell(row, rightPos);
            if (rightCell != '.') {
                blockingPieces.add(rightCell);
            }
        }
    } else {
        int col = piece.getCol();

        if (piece.getRow() > 0) {
            char upCell = board.getCell(piece.getRow() - 1, col);
            if (upCell != '.') {
                blockingPieces.add(upCell);
            }
```

```
        }

        int bottomPos = piece.getRow() + piece.getSize();
        if (bottomPos < board.getRows()) {
            char downCell = board.getCell(bottomPos, col);
            if (downCell != '.') {
                blockingPieces.add(downCell);
            }
        }
    }

    return blockingPieces;
  }

  private int calculateMobilityScore(Board board, Piece p, Set<Character> directBlockers)
{
    int mobilityScore = 0;
    Map<Character, Piece> pieces = board.getPieces();

    int targetMobility = calculatePieceMobility(board, p);
    mobilityScore += targetMobility * 2;

    for (char blockerId : directBlockers) {
        Piece blocker = pieces.get(blockerId);
        if (blocker == null) continue;

        int blockerMobility = calculatePieceMobility(board, blocker);
        mobilityScore += blockerMobility;
    }

    return mobilityScore;
  }

  private int calculatePieceMobility(Board board, Piece piece) {
    int mobility = 0;

    if (piece.isHorizontal()) {
        int row = piece.getRow();

        for (int c = piece.getCol() - 1; c >= 0; c--) {
            if (board.getCell(row, c) == '.') {
                mobility++;
            } else {
                break;
            }
        }

        for (int c = piece.getCol() + piece.getSize(); c < board.getCols(); c++) {
            if (board.getCell(row, c) == '.') {
                mobility++;
            } else {
                break;
            }
        }
    } else {
        int col = piece.getCol();
```

```
            for (int r = piece.getRow() - 1; r >= 0; r--) {
                if (board.getCell(r, col) == '.') {
                    mobility++;
                } else {
                    break;
                }
            }

            for (int r = piece.getRow() + piece.getSize(); r < board.getRows(); r++) {
                if (board.getCell(r, col) == '.') {
                    mobility++;
                } else {
                    break;
                }
            }
        }

        return mobility;
    }
}
```

5.2.2. Manhattan.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.HashSet;
import java.util.Set;

public class Manhattan implements Heuristic {
    public int calculate(State state) {
        Board board = state.getBoard();
        Piece p = board.getPieces().get('P');
        int er = board.getExitRow(), ec = board.getExitCol();

        if (p.isHorizontal()) {
            int pmr = p.getRow();
            int pmc = p.getCol() + (p.getSize() - 1) / 2;

            return Math.abs(ec - pmc) + Math.abs(er - pmr);
        }
        else {
            int pmr = p.getRow() + (p.getSize() - 1) / 2;
            int pmc = p.getCol();

            return Math.abs(ec - pmc) + Math.abs(er - pmr);
        }
    }
}
```

5.2.3. Pattern.java

```
package com.java.searching.heuristic;

import com.java.model.Board;
import com.java.model.Piece;
import com.java.searching.State;
import java.util.*;
import java.io.*;

public class Pattern implements Heuristic {
    private static Map<String, Integer> patternDatabase = new HashMap<>();

    private static List<Set<Character>> patterns = new ArrayList<>();

    static {
        initializePatterns();
        loadPatternDatabase();
    }

    public int calculate(State state) {
        int total = 0;
        Board board = state.getBoard();

        for (Set<Character> pattern : patterns) {
            String patternHash = extractPatternHash(board, pattern);

            if (patternDatabase.containsKey(patternHash)) {
                total += patternDatabase.get(patternHash);
            } else {
                total += calculateBaseHeuristic(board, pattern);
            }
        }
        return total;
    }

    private static void initializePatterns() {
        patterns.add(new HashSet<>(Arrays.asList('P')));
        Set<Character> exitRegionPattern = new HashSet<>();
        exitRegionPattern.add('P');
        patterns.add(exitRegionPattern);
        Set<Character> cornerPattern = new HashSet<>();
        patterns.add(cornerPattern);
    }

    private static String getDbFilePath() {
        String homeDir = System.getProperty("user.home");
        return homeDir + File.separator + "rushHourPatternDB.ser";
    }

    private static void loadPatternDatabase() {
        try {
            File dbFile = new File(getDbFilePath());
            if (dbFile.exists()) {
                ObjectInputStream ois = new ObjectInputStream(new FileInputStream(dbFile));
```

```
            patternDatabase = (Map<String, Integer>) ois.readObject();
            ois.close();
            System.out.println("Loaded pattern database with " + patternDatabase.size() + "
entries");
        } else {
            System.out.println("No pattern database found, using fallback heuristics");
        }
    } catch (Exception e) {
        System.out.println("Error loading pattern database: " + e.getMessage());
    }
  }

  private String extractPatternHash(Board board, Set<Character> pattern) {
    StringBuilder hash = new StringBuilder();

    Piece p = board.getPieces().get('P');
    int er = board.getExitRow();
    int ec = board.getExitCol();

    if (pattern.contains('P')) {
        hash.append("P:")
            .append(p.getRow()).append(",")
            .append(p.getCol()).append(",")
            .append(p.getSize()).append(",")
            .append(p.isHorizontal() ? "H" : "V").append(";");
    }

    hash.append("K:").append(er).append(",").append(ec).append(";");

    if (pattern == patterns.get(1)) {
        if (p.isHorizontal()) {
            int row = p.getRow();
            for (int c = 0; c < board.getCols(); c++) {
                char cell = board.getCell(row, c);
                if (cell != '.' && cell != 'P' && !pattern.contains(cell)) {
                    pattern.add(cell);
                }
            }
        }
        else {
            int col = p.getCol();
            for (int r = 0; r < board.getRows(); r++) {
                char cell = board.getCell(r, col);
                if (cell != '.' && cell != 'P' && !pattern.contains(cell)) {
                    pattern.add(cell);
                }
            }
        }
    }

    if (pattern == patterns.get(2)) {
        if (board.getCell(0, 0) != '.') pattern.add(board.getCell(0, 0));
        if (board.getCell(0, board.getCols()-1) != '.') pattern.add(board.getCell(0,
board.getCols()-1));
        if (board.getCell(board.getRows()-1, 0) != '.')
pattern.add(board.getCell(board.getRows()-1, 0));
```

```
        if (board.getCell(board.getRows()-1, board.getCols()-1) != '.')
            pattern.add(board.getCell(board.getRows()-1, board.getCols()-1));
    }

    for (char pieceId : pattern) {
        if (pieceId == 'P') continue;

        Piece piece = board.getPieces().get(pieceId);
        if (piece != null) {
            hash.append(pieceId).append(":")
                .append(piece.getRow()).append(",")
                .append(piece.getCol()).append(",")
                .append(piece.getSize()).append(",")
                .append(piece.isHorizontal() ? "H" : "V").append(";");
        }
    }
    return hash.toString();
}

private int calculateBaseHeuristic(Board board, Set<Character> pattern) {
    Piece p = board.getPieces().get('P');
    int er = board.getExitRow();
    int ec = board.getExitCol();

    if (pattern == patterns.get(0)) {
        if (p.isHorizontal()) {
            int frontCol = p.getCol() + p.getSize() - 1;
            return Math.abs(ec - frontCol);
        } else {
            int frontRow = p.getRow() + p.getSize() - 1;
            return Math.abs(er - frontRow);
        }
    }
    else if (pattern == patterns.get(1)) {
        Set<Character> blockers = new HashSet<>();

        if (p.isHorizontal()) {
            int row = p.getRow();
            int frontCol = p.getCol() + p.getSize() - 1;
            int start = Math.min(frontCol, ec);
            int end = Math.max(frontCol, ec);

            for (int c = start + 1; c < end; c++) {
                char cell = board.getCell(row, c);
                if (cell != '.') blockers.add(cell);
            }
        } else {
            int col = p.getCol();
            int frontRow = p.getRow() + p.getSize() - 1;
            int start = Math.min(frontRow, er);
            int end = Math.max(frontRow, er);

            for (int r = start + 1; r < end; r++) {
                char cell = board.getCell(r, col);
                if (cell != '.') blockers.add(cell);
            }
```

```
            }

            return blockers.size() * 2;
        }
        else if (pattern == patterns.get(2)) {
            int penalty = 0;
            int rows = board.getRows();
            int cols = board.getCols();

            if (board.getCell(0, 0) != '.' && board.getCell(0, 1) != '.' &&
                board.getCell(1, 0) != '.') {
                penalty += 1;
            }

            if (board.getCell(0, cols-1) != '.' && board.getCell(0, cols-2) != '.' &&
                board.getCell(1, cols-1) != '.') {
                penalty += 1;
            }

            if (board.getCell(rows-1, 0) != '.' && board.getCell(rows-1, 1) != '.' &&
                board.getCell(rows-2, 0) != '.') {
                penalty += 1;
            }

            if (board.getCell(rows-1, cols-1) != '.' && board.getCell(rows-1, cols-2) != '.' &&
                board.getCell(rows-2, cols-1) != '.') {
                penalty += 1;
            }
            return penalty;
        }

        return 0;
    }

    public static void buildPatternDatabase() {
        System.out.println("Building pattern database - this may take a while...");

        try {
            ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(getDbFilePath()));
            oos.writeObject(patternDatabase);
            oos.close();
            System.out.println("Saved pattern database with " + patternDatabase.size() + "
entries");
        } catch (IOException e) {
            System.out.println("Error saving pattern database: " + e.getMessage());
        }
    }

    private static class PatternState implements Serializable {
        Map<Character, Piece> pieces;
        int er, ec;

        public PatternState(Map<Character, Piece> pieces, int er, int ec) {
            this.pieces = pieces;
            this.er = er;
```

```
            this.ec = ec;
        }
    }
}
```

5.3. Pathfinding Alternatif

Disini, program juga mengimplementasikan Algoritma pathfinding alternatif;

5.3.1. IDAStarSolver.java

Iterative Deepening A* (IDA* / IDA-Star) adalah varian dari algoritma A* yang memadukan ketelitian A* dengan konsumsi memori serendah *depth-first search*. Ia tetap *informed search*—menjadi "cerdas" berkat heuristik—tetapi menelusuri ruang‑status secara bertahap memakai batas (*threshold*) yang kian diperlebar.

Dalam konteks Rush Hour, IDA* menjadi kompromi logis: ia mengekspansi jauh lebih sedikit simpul daripada UCS yang "buta", tetap menemukan jalur minimum langkah layaknya A*, namun menahan kebutuhan memori agar tidak meledak saat puzzle semakin padat.

```
package com.java.searching;

import com.java.checker.PuzzleChecker;
import com.java.exception.InvalidConfigurationException;
import java.util.*;

public class IDAStarSolver implements SearchStrategy {
    public SolverResult solve(State start) {
        long t0 = System.currentTimeMillis();
        int threshold = start.heuristic();
        long nodes = 0;

        List<Move> path = new ArrayList<>();

        int iteration = 0;
        int maxIterations = 100; // Safety limit for extremely difficult puzzles

        while (iteration++ < maxIterations) {
            path.clear();

            Map<String, Integer> bestCost = new HashMap<>();

            Result result = new Result();
            result.nodesExpanded = nodes;

            boolean found = dfs(start, 0, threshold, path, bestCost, result);
            nodes = result.nodesExpanded;
```

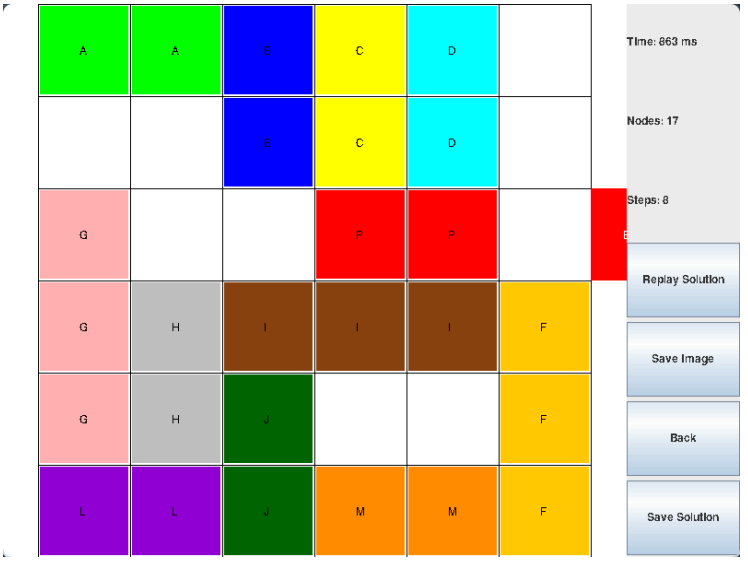
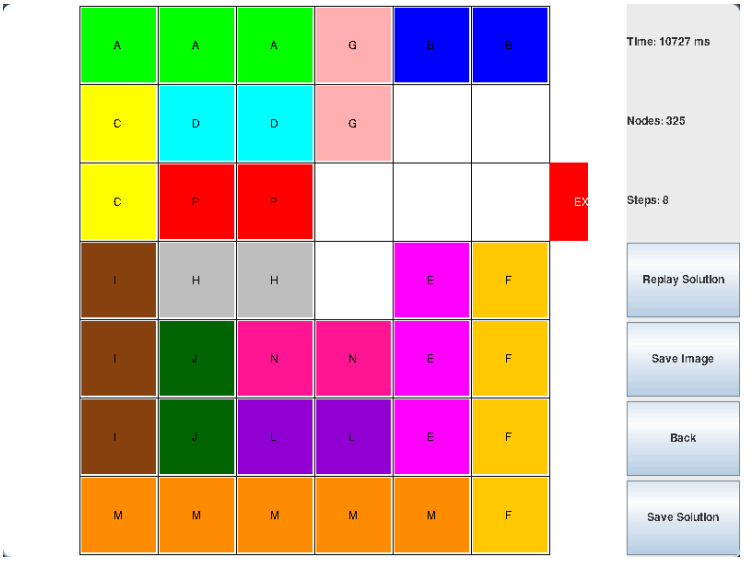
```
            if (found) {
                return new SolverResult(path, nodes, System.currentTimeMillis() - t0);
            }

            if (result.nextThreshold == Integer.MAX_VALUE) {
                return new SolverResult(new ArrayList<>(), nodes, System.currentTimeMillis() -
t0);
            }

            threshold = result.nextThreshold;

            System.out.println("IDA* iteration " + iteration + ": increased threshold to " +
threshold);
        }

        // If we hit iteration limit, return best effort
        return new SolverResult(new ArrayList<>(), nodes, System.currentTimeMillis() - t0);
    }

    private static class Result {
        int nextThreshold = Integer.MAX_VALUE;
        long nodesExpanded = 0;
    }

    private boolean dfs(State state, int g, int threshold, List<Move> path,
                        Map<String, Integer> bestCost, Result result) {
        result.nodesExpanded++;

        int h = state.heuristic();
        int f = g + h;

        // Prune if f-value exceeds threshold
        if (f > threshold) {
            result.nextThreshold = Math.min(result.nextThreshold, f);
            return false;
        }

        // Check for goal state
        boolean isGoal;
        try {
            isGoal = PuzzleChecker.checkSolved(state.getBoard());
        } catch (InvalidConfigurationException e) {
            isGoal = false;
        }

        if (isGoal) {
            return true;
        }

        String stateKey = state.toString();

        Integer previousCost = bestCost.get(stateKey);
        if (previousCost != null && previousCost <= g) {
            return false;
        }
```

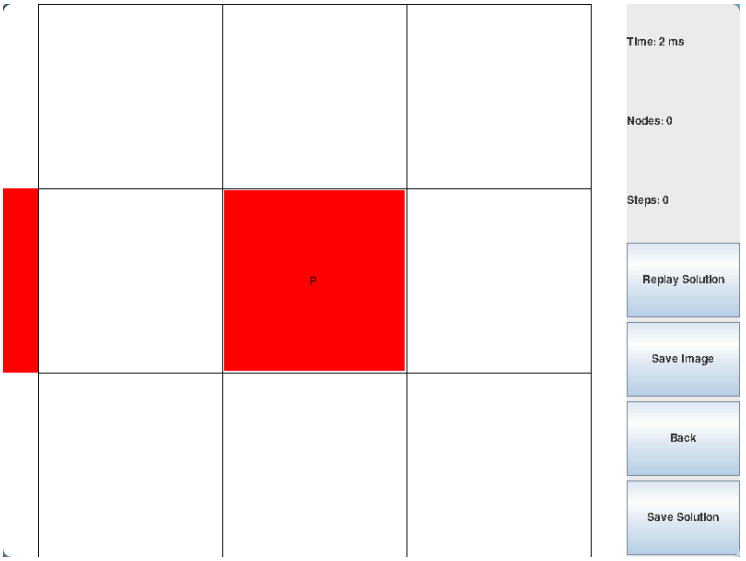
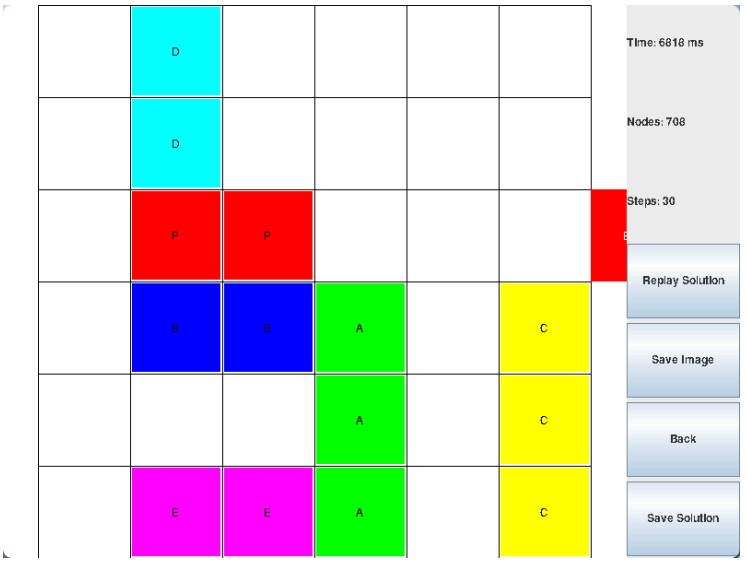
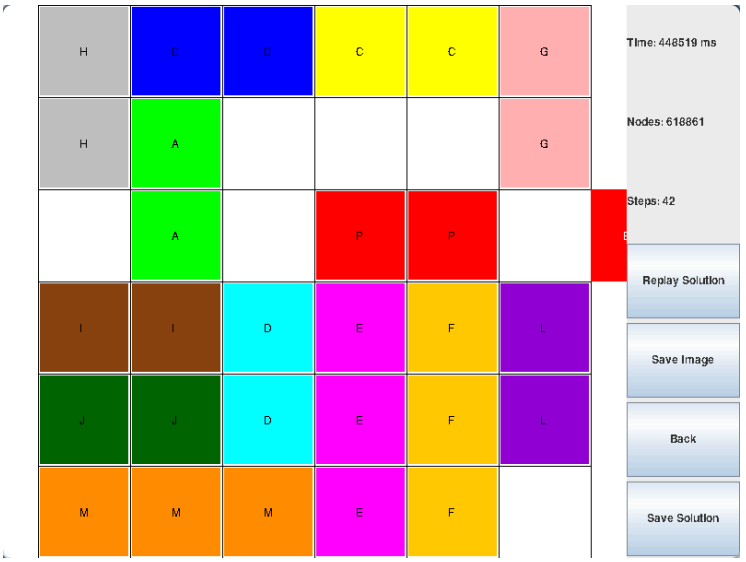
```
        bestCost.put(stateKey, g);

        List<Move> successors = state.successors();
        successors.sort(Comparator.comparingInt(move -> g + 1 + move.next.heuristic()));

        // Try each successor
        for (Move move : successors) {
            path.add(move);

            if (dfs(move.next, g + 1, threshold, path, bestCost, result)) {
                return true;
            }

            path.remove(path.size() - 1);
        }

        return false;
    }
}
```

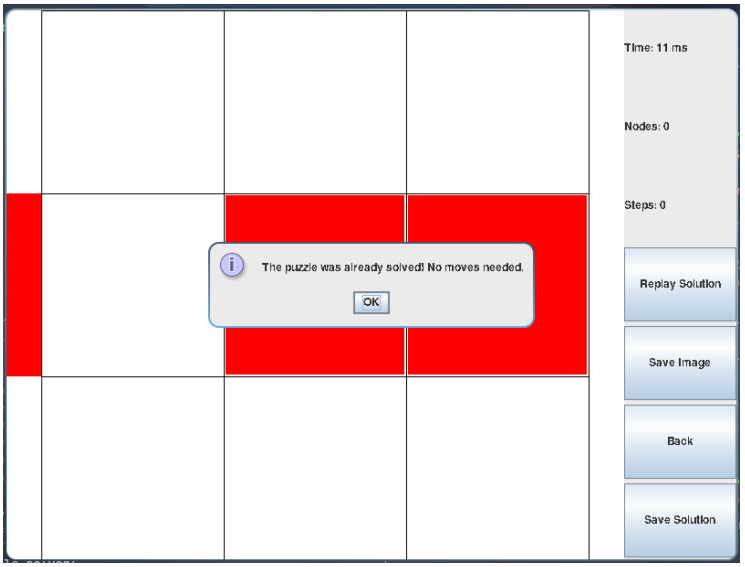
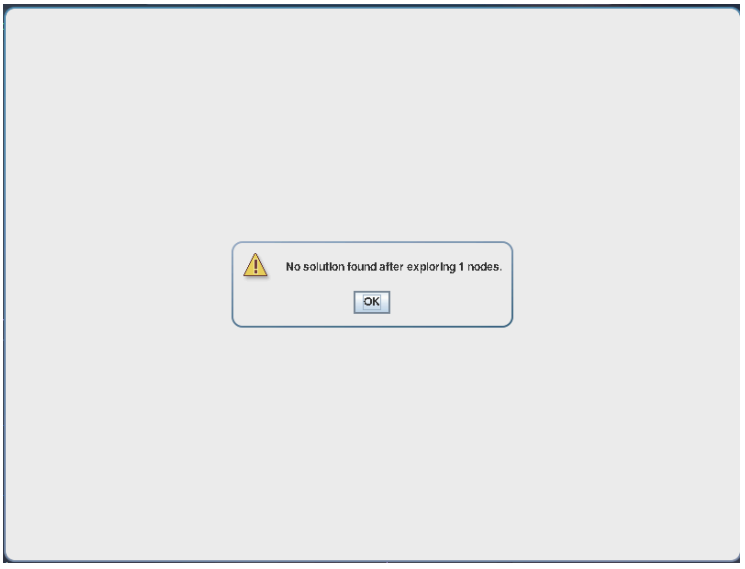
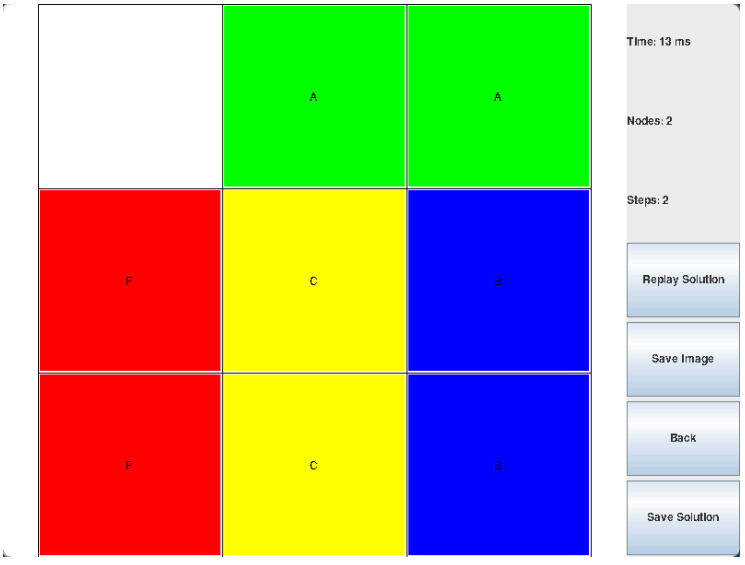
BAB VI
EKSPERIMEN DAN ANALISIS

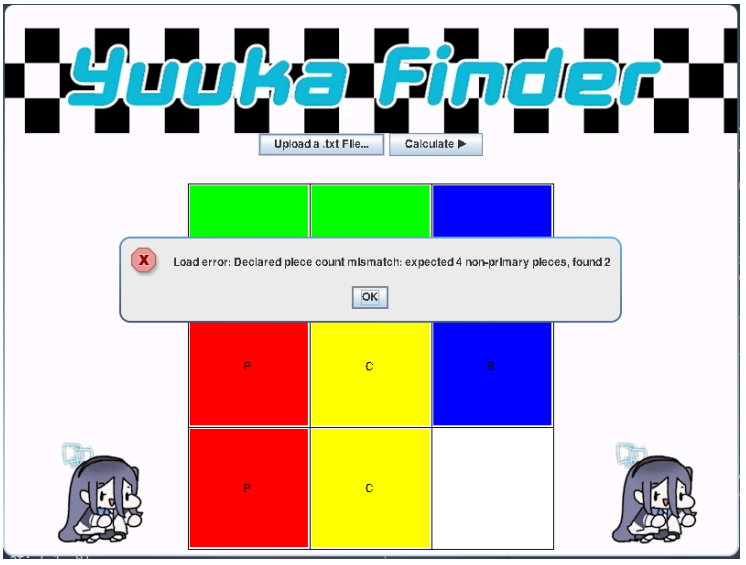
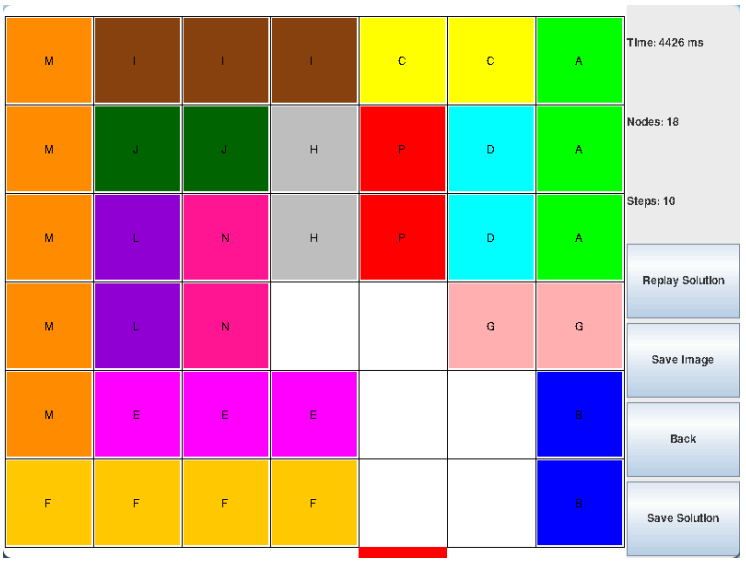
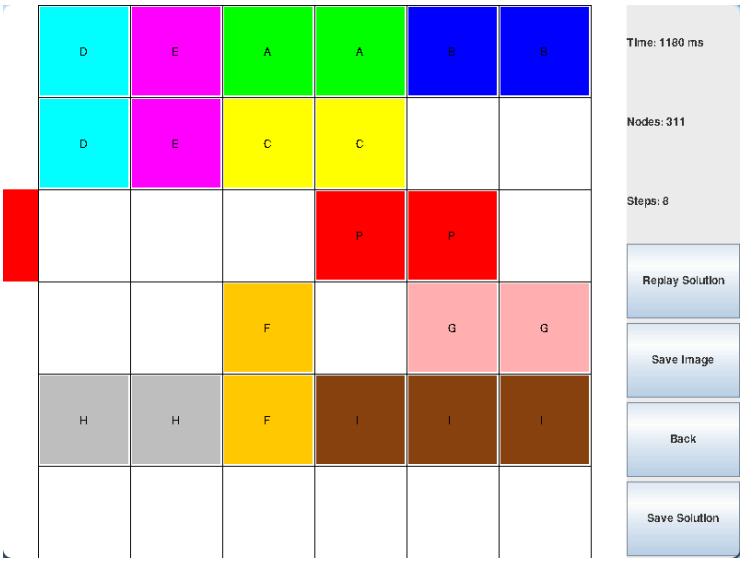
6.1. Pengujian Program

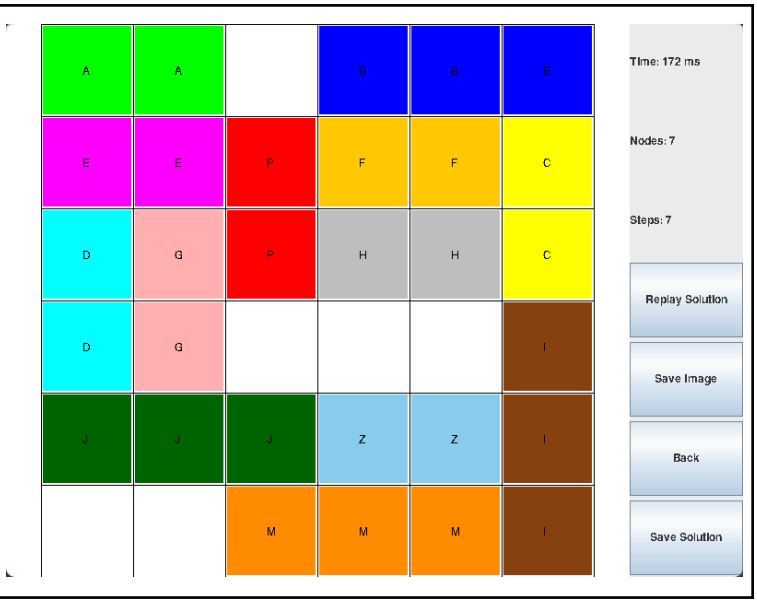
Berikut adalah beberapa *test case* yang dicoba pada program, beserta hasil *output* (keluaran) pada GUI berupa screenshot:

No	INPUT			OUTPUT
	Config file	**Algoritma**	***Heuristics***	
1.	6 6 11 AAB..F ..BCDF G**PP**CDF**K** GH.III GHJ... LLJMM.	GBFS	EBlocking	
2.	7 6 13 AAABB. CDD..F C**PP**GEF**K** .HHGEF IJNNEF IJLL.. IMMMMM	A*	Pattern Database	

3.	3 3 0 ... **K.P.** ...	A*	Composite	
4.	6 6 5 ....**PP**A..**K** .BBA.C .D.A.C .DEE.C	UCS	-	
5.	6 6 12 .ABBCC .AD... **PP**DEFG**K** HIIEFG HJJEFL MMM..L	IDA*	Manhattan Distance	

6.	3 3 0 ... **K.PP** ...	IDA*	Composite	
7.	3 3 3 AAA **PP**B**K** CCB	A*	Manhattan Distance	
8.	3 3 3 **K** AAB **P**CB **P**C.	UCS	-	

9.	3 3 4 AA. BB.**K** **PP**.	GBFS	Pattern Database	
10.	6 7 13 III.CCA MJJH**P**DA MLNH**P**DA MLNGG.B M.EEE.B M.FFFF. **K**	IDA*	Pattern Database	
11.	6 6 9 AA.BB. CC.... **K**DEF**PP**. DEFGGG HHIIIJ .....J	A*	Pattern Database	

12.	6 6 12 **K** AABBBC DEEFFC DG**P**HHI .G**P**..I JJJZZI ...MMM	GBFS	EBlocking	

6.2. Analisis Hasil Percobaan

Analisis kompleksitas algoritma yang digunakan dibagi menjadi beberapa bagian. Hal ini dilakukan karena tipe pathfinding yang dipakai mempengaruhi hasil node, program search time, dan steps yang diambil. Pembagian analisis kompleksitas juga mempermudah dan membuat lebih jelas perhitungan kompleksitas algoritma yang digunakan. Berikut adalah analisis kompleksitas per bagian dari algoritma program yang dibuat:

6.2.1. Analisis Kompleksitas Algoritma UCS

Algoritma Uniform Cost Search mengekspansi simpul berdasarkan biaya kumulatif *g(n)*; dengan asumsi semua langkah bernilai non-negatif, perilakunya ekuivalen dengan Breadth-First Search ketika setiap gerakan memiliki bobot 1.

- Branching factor (b) : rata-rata banyaknya gerakan legal dari satu konfigurasi papan
- Kedalaman solusi (d) : jumlah langkah minimum sampai mobil utama keluar

Waktu terburuk: $O(b^d)$ Ruang Terburuk: $O(b^d)$

Karena UCS tidak memiliki petunjuk arah (*uninformed / blind-search*), ia mem-visit semua simpul pada kedalaman *d* sebelum beralih ke *d+1*. Pada puzzle Rush Hour dengan cabang luas atau solusi dalam, percobaan menunjukkan:

- jumlah simpul yang dibangkitkan jauh lebih banyak daripada A*
- penggunaan memori melonjak (priority-queue berisi hampir seluruh frontier)

- waktu eksekusi menjadi paling lama di antara ketiga algoritma, walau solusi yang diperoleh tetap optimal.

6.2.2. Analisis Kompleksitas Algoritma GBFS

Greedy Best-First Search (GBFS) hanya melihat heuristik *h(n)* di eksperimen, jumlah kendaraan yang memblokir pintu keluar—tanpa mempertimbangkan biaya lampau (*g(n)=0* untuk semua *n*).

- Kedalaman solusi tercapai (m) : langkah pada jalur pertama yang dianggap menuju goal.

Waktu rata-rata: $O(b^m)$ Ruang rata-rata: $O(b^m)$

Nilai *m* sering kali < *d*, sehingga GBFS *praktis* jauh lebih cepat dan hemat memori daripada UCS. Hasil percobaan memperlihatkan:

- waktu penyelesaian tercepat, karena fokus langsung ke arah goal,
- panjang solusi cenderung lebih besar—kadang signifikan—dibanding UCS maupun A*,
- risiko terjebak pada jalur buntu meningkat jika heuristik kurang informatif atau tidak *admissible*.

Dengan kata lain, GBFS mengorbankan optimalitas demi efisiensi waktu, cocok untuk skenario di mana respon cepat lebih penting daripada jalur terpendek.

6.2.3. Analisis Kompleksitas Algoritma A* (A-star)

Algoritma A* (A-star) mengevaluasi simpul menggunakan fungsi $f(n) = g(n) + h(n)$ yang menyeimbangkan biaya nyata dan estimasi sisa biaya. Jika *h(n)* admissible dan konsisten, A* dijamin menemukan jalur optimal dengan eksplorasi yang lebih terarah.

Waktu terburuk: $O(b^d)$ Ruang terburuk: $O(b^d)$

Namun—berkat heuristik yang baik—jumlah simpul nyata yang dikunjungi jauh di bawah UCS. Dari percobaan:

- A* selalu menghasilkan solusi dengan jumlah langkah minimum, setara UCS;
- penggunaan memori dan waktu rata-rata paling efisien di antara metode yang juga menjamin optimalitas

- performa tetap bergantung pada kualitas heuristik *h(n)*. Semakin dekat taksirannya ke jarak sesungguhnya, semakin kecil simpul yang diekspansi.

LAMPIRAN

Checklist Spesifikasi Program

Tabel 7.1. Tabel *checklist* spesifikasi program

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	☑	☐
2	Program berhasil dijalankan	☑	☐
3	Solusi yang diberikan program benar dan mematuhi peraturan 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] GUI	☑	☐
8	Program dan laporan dibuat (kelompok) sendiri	☑	☐

Pembagian Tugas

Tabel 7.2. Tabel pembagian tugas

Tugas	NIM
Github	13523123, 13523147
Main Pathfinding Calculation (A*, GBFS, UCS)	13523123, 13523147
[BONUS] GUI	13523123, 13523147
[BONUS] Heuristic	13523147
[BONUS] Alternative Pathfinding	13523147
Laporan	13523123, 13523147

Repository

Github: **Repository**

(raw link: https://github.com/susTuna/Tucil3_13523123_13523147)

Yuuka Hayase:

ini kenapa masih disini :v

DAFTAR PUSTAKA

Munir, Rinaldi. 2025. Bagian 1: BFS, DFS, UCS, Greedy Best First Search. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf. Diakses pada 20 Mei 2025.

Munir, Rinaldi. 2025. Bagian 2: Algoritma A*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf. Diakses pada 20 Mei 2025.