

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



Disusun oleh :

Shannon Aurellius Anastasya Lie (13523019)

Naomi Risaka Sitorus (13523122)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2025**

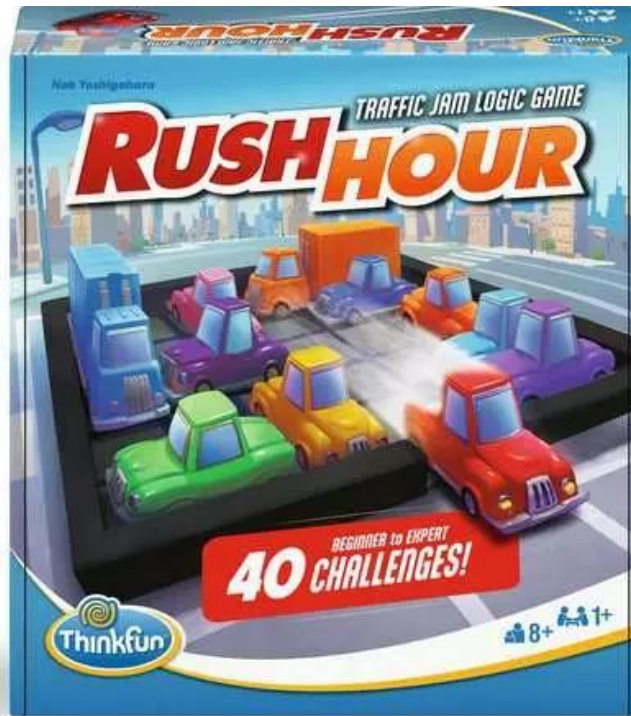
## DAFTAR ISI

<b>BAB I DESKRIPSI PERSOALAN.....</b>	<b>4</b>
<b>BAB II LANDASAN TEORI .....</b>	<b>8</b>
2.1. Algoritma Pathfinding Greedy Best First Search (GBFS).....	8
2.2. Algoritma Pathfinding UCS (Uniform Cost Search) .....	8
2.3. Algoritma Pathfinding A* (A-Star) .....	8
2.4. Algoritma Pathfinding IDS (Iterative Deepening Search) (Bonus) .....	8
2.5. Alur Kerja Program Rush Hour Puzzle Solver dengan Algoritma Pathfinding .....	9
<b>BAB III IMPLEMENTASI ALGORITMA .....</b>	<b>12</b>
3.1. Solver .....	12
3.2. Algoritma Pathfinding GBFS (Greedy Best First Search).....	12
3.3. Algoritma Pathfinding UCS (Uniform Cost Search) .....	13
3.4. Algoritma Pathfinding A* (A-Star) .....	13
3.5. Algoritma Pathfinding IDS (Iterative Deepening Search) (Bonus) .....	14
3.6. Jenis Heuristik (Bonus).....	15
3.7. Board.....	16
3.8. Input Parser .....	17
3.9. Main dengan GUI (Bonus).....	18
<b>BAB IV SOURCE CODE PROGRAM .....</b>	<b>19</b>
4.1. Solver.java .....	19
4.2. GBFSSolver.java .....	20
4.3. UCSSolver.java.....	21
4.4. AStarSolver.java .....	23
4.5. IDSSolver.java .....	25
4.6. Heuristic.java .....	27
4.7. Board.java .....	31
4.8. InputParser.java .....	39
4.9. GUIApp.java.....	46
<b>BAB V PENGUJIAN MASUKAN DAN LUARAN PROGRAM.....</b>	<b>57</b>
5.1. Test Case 1 .....	57
5.2. Test Case 2.....	60
5.3. Test Case 3 .....	64
5.4. Test Case 4 .....	68
5.5. Test Case 5 .....	72
5.6. Test Case 6.....	75
5.7. Test Case 7 .....	78
5.8. Test Case 8.....	81

5.9. Test Case 9 .....	82
5.10. Test Case 10 .....	83
5.11. Test Case 11 .....	84
5.12. Test Case 12 .....	87
5.13. Test Case 13 .....	90
5.14. Test Case 14 .....	93
5.15. Test Case 15 .....	96
5.16. Test Case 16 .....	99
5.17. Test Case 17 .....	102
5.18. Test Case 18 .....	105
5.19. Test Case 19 .....	108
5.20. Test Case 20 .....	111
5.21. Test Case 21 .....	114
5.22. Test Case 22 .....	117
5.23. Test Case 23 .....	120
5.24. Test Case 24 .....	123
5.25. Test Case 25 .....	126
<b>BAB VI ANALISIS ALGORITMA .....</b>	<b>130</b>
6.1. Algoritma Pathfinding Greedy Best First Search .....	130
6.2. Algoritma Pathfinding UCS (Uniform Cost Search) .....	131
6.3. Algoritma Pathfinding A* (A-Star) .....	132
6.4. Algoritma Pathfinding IDS (Iterative Deepening Search) .....	133
6.5. Algoritma Heuristik .....	134
<b>BAB VII KESIMPULAN DAN SARAN .....</b>	<b>136</b>
7.1. Kesimpulan .....	136
7.2. Saran .....	136
7.3. Komentar .....	137
7.4. Refleksi .....	137
<b>DAFTAR PUSTAKA .....</b>	<b>138</b>
<b>LAMPIRAN .....</b>	<b>139</b>

## BAB I

### DESKRIPSI PERSOALAN



**Gambar 1.** Rush Hour Puzzle

(Sumber: <https://www.thinkfun.com/en-US/products/educational-games/rush-hour-76582>)

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

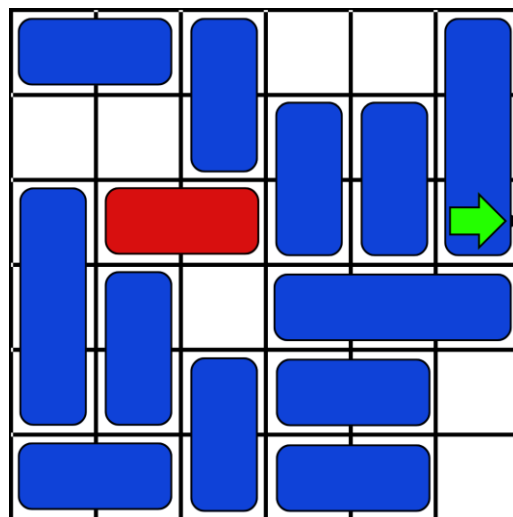
Komponen penting dari permainan Rush Hour terdiri dari:

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

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

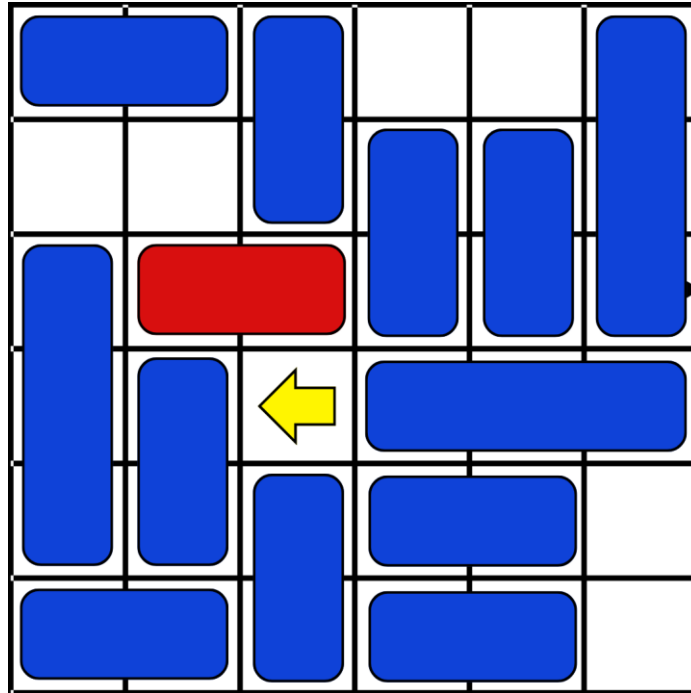
Ilustrasi kasus :

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

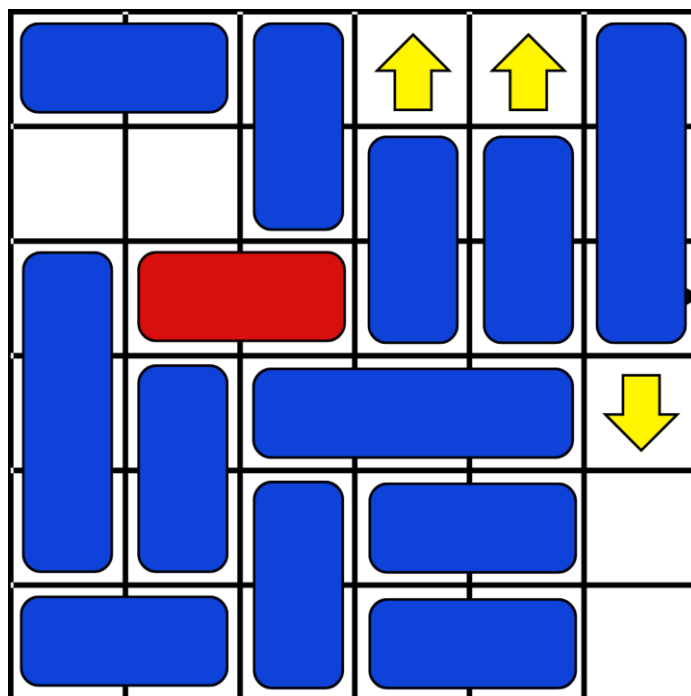


Gambar 2. Awal Permainan Game Rush Hour

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

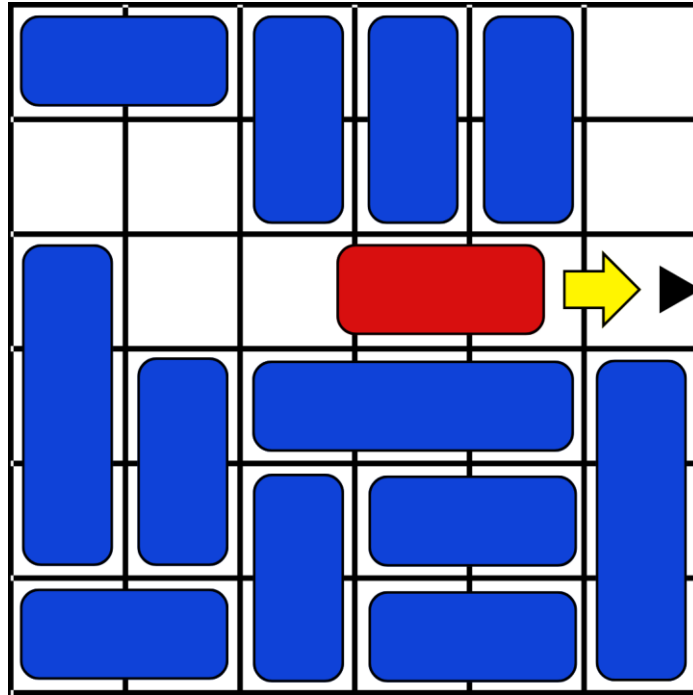


Gambar 3. Gerakan Pertama Game Rush Hour



Gambar 4. Gerakan Kedua Game Rush Hour

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



Gambar 5. Pemain Menyelesaikan Permainan

Agar lebih jelas, silahkan amati video cara bermain berikut: [The New Rush Hour by ThinkFun!](#)

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

## BAB II

### LANDASAN TEORI

#### 2.1. Algoritma *Pathfinding Greedy Best First Search* (GBFS)

*Greedy Best First Search* (GBFS) merupakan algoritma pencarian jalur (*pathfinding*) yang mengandalkan pendekatan heuristik untuk memperkirakan jarak dari simpul saat ini ke tujuan. Pada setiap langkah, algoritma ini memilih simpul yang tampak paling dekat dengan tujuan berdasarkan nilai heuristik  $h(n)$ , tanpa memperhatikan biaya dari simpul awal ke simpul tersebut. Hal ini membuat GBFS seringkali lebih cepat dibanding algoritma lain, tetapi tidak menjamin solusi optimal karena dapat mengabaikan jalur dengan total biaya yang lebih rendah. GBFS cocok untuk kasus ketika kecepatan menjadi prioritas utama dibandingkan optimalisasi hasil.

#### 2.2. Algoritma *Pathfinding UCS* (*Uniform Cost Search*)

*Uniform Cost Search* (UCS) adalah algoritma pencarian yang menjamin hasil optimal dengan memprioritaskan eksplorasi simpul berdasarkan biaya kumulatif terkecil dari titik awal ke simpul tersebut, yang dilambangkan dengan  $g(n)$ . UCS tidak menggunakan fungsi heuristik, sehingga berbeda dengan algoritma seperti A\* atau GBFS. Karena UCS selalu memilih jalur termurah terlebih dahulu, algoritma ini sangat efektif pada graf berbobot untuk menemukan jalur terpendek. Kelemahannya, UCS bisa menjadi lambat jika banyak jalur memiliki biaya serupa atau jika grafnya sangat besar.

#### 2.3. Algoritma *Pathfinding A\** (*A-Star*)

A\* (*A-Star*) merupakan salah satu algoritma pathfinding yang paling populer karena menggabungkan keuntungan dari UCS dan GBFS. A\* menggunakan fungsi evaluasi  $f(n) = g(n) + h(n)$ , yaitu jumlah antara biaya dari awal ke simpul saat ini ( $g(n)$ ) dan estimasi biaya dari simpul saat ini ke tujuan ( $h(n)$ ). Dengan demikian, A\* mempertimbangkan baik akumulasi biaya sejauh ini maupun prediksi ke depan. Jika heuristik yang digunakan bersifat *admissible* (tidak melebihi-lebihkan biaya sesungguhnya), A\* menjamin solusi optimal. Efisiensi dan fleksibilitas A\* menjadikannya algoritma utama dalam banyak aplikasi navigasi dan game AI.

#### 2.4. Algoritma *Pathfinding IDS* (*Iterative Deepening Search*) (Bonus)

Iterative Deepening Search (IDS) adalah pendekatan pencarian jalur yang menggabungkan keunggulan pencarian *depth-first* dan *breadth-first*. IDS bekerja dengan menjalankan *depth-limited search* secara berulang-ulang dengan batas kedalaman yang terus meningkat, dimulai dari 0 hingga ditemukan solusi. Meskipun tampak redundan karena simpul atas dieksplorasi ulang setiap iterasi, IDS memiliki keunggulan dalam efisiensi memori seperti pencarian *depth-first*, namun tetap menjamin solusi optimal seperti *breadth-first search* jika semua langkah memiliki



bobot yang sama. IDS sangat cocok digunakan ketika ruang pencarian sangat besar dan tidak diketahui kedalamannya.

## 2.5. Alur Kerja Program Rush Hour Puzzle Solver dengan Algoritma Pathfinding

Pencarian solusi dari permainan Rush Hour dapat dilakukan dengan pendekatan algoritma *pathfinding* sebagai berikut.

1. Program meminta masukan berupa konfigurasi permainan/*test case*.

Pengguna diminta untuk memasukkan nama file konfigurasi permainan berekstensi .txt dengan *browsing file explorer* di GUI. Apabila file masukan berhasil dibaca, program akan melakukan validasi terhadap format serta konten dalam file tersebut. Baris pertama harus berisi nilai A dan B dengan  $A \times B$  mewakili ukuran papan permainan. Baris kedua harus berisi nilai N yang mewakili jumlah kendaraan (*piece*) bukan *primary piece* yang ada.

Baris selanjutnya dalam file masukan merupakan isi papan permainan yang jumlah *piece*-nya harus berjumlah N buah, memiliki satu buah *primary piece* yang ditandai dengan huruf P, dan memiliki satu pintu keluar yang ditandai dengan huruf K. Setiap *piece* harus berbentuk linear, yakni horizontal atau vertikal, dan berukuran 1 x 2 atau 1 x 3. Setiap huruf hanya dapat merepresentasikan satu *piece* dan jumlah *piece* dalam satu permainan dijamin tidak lebih dari 24 buah. Seluruh *piece* disimpan dalam *board* permainan.

2. Pengguna memilih algoritma *pathfinding* yang ingin digunakan untuk pencarian solusi.

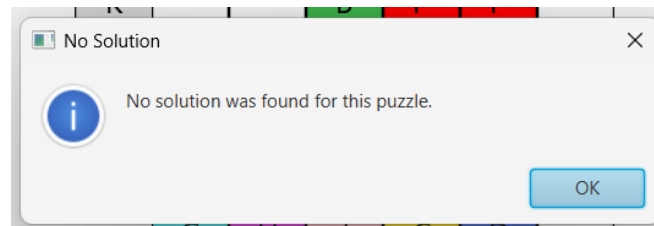
Pilihan algoritma tersebut terdiri dari 4 algoritma yaitu 3 algoritma *pathfinding* wajib berupa UCS (*Uniform Cost Search*), GBFS (*Greedy Best First Search*), dan A\* (*A-Star*), serta 1 algoritma *pathfinding* bonus yaitu IDS (*Iterative Deepening Search*) yang merupakan bonus. Khusus untuk IDS, pengguna juga diminta untuk memasukkan kedalaman yang ingin ditelusuri. Heuristik yang dipakai juga dapat dipilih yaitu berupa *Blocking Pieces Count*, *Blocking Pieces With Movability*, dan *Distance-to-Exit* yang merupakan bonus. Hal ini akan berpengaruh pada pencarian solusi jika menggunakan algoritma GBFS atau A\*. Pengguna juga dapat memilih bentuk tampilan solusi, yakni paginasi atau animasi.

3. Program menjalankan algoritma *pathfinding* untuk mencari solusi permainan.

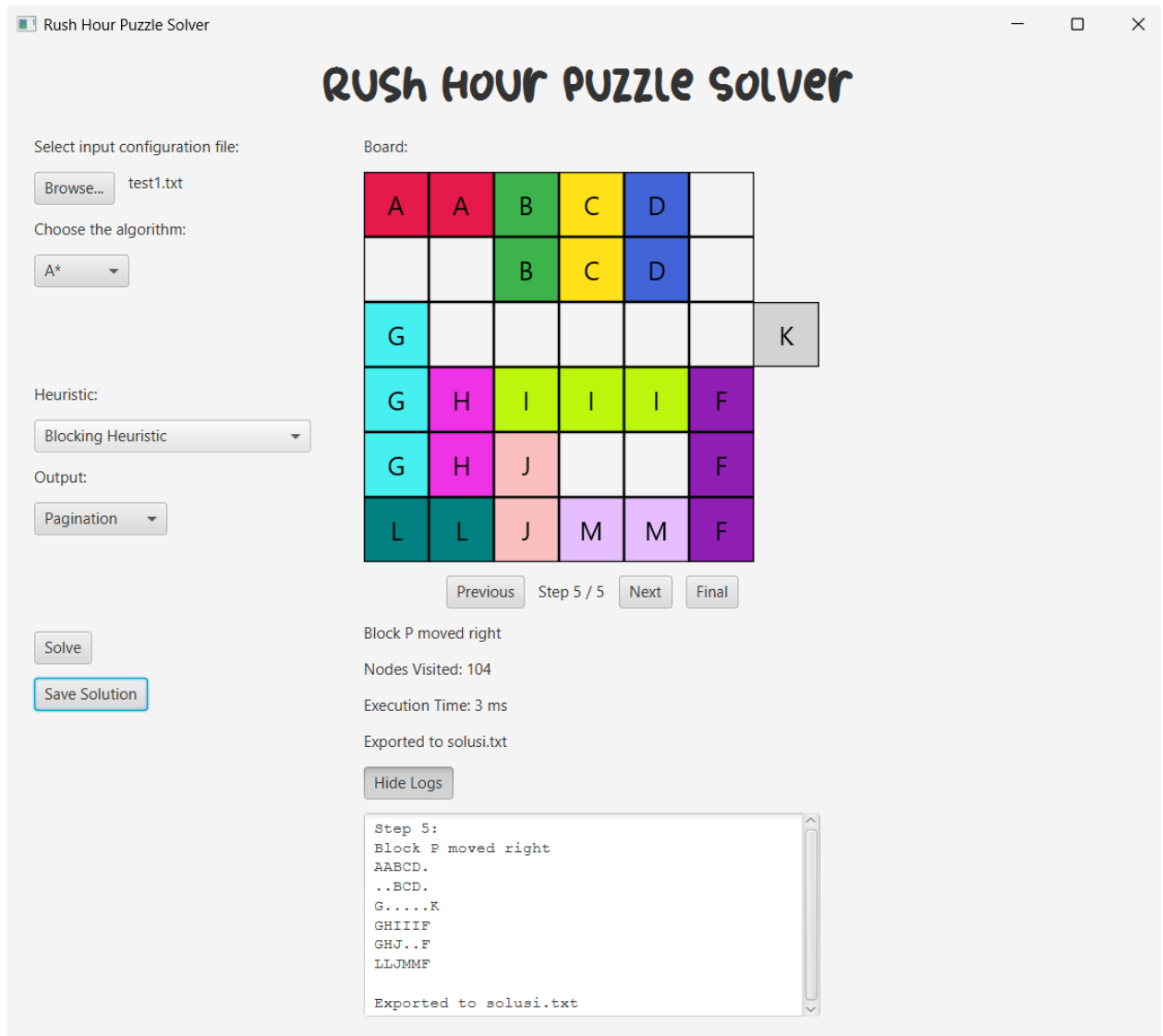
Program akan mulai dijalankan jika pengguna sudah mengisi seluruh masukan yang diperlukan dan menekan tombol *Solve* pada GUI. Program akan menghitung  $f(n)$ ,  $g(n)$ , dan  $h(n)$  sesuai algoritma yang dipilih: UCS menggunakan  $g(n)$ , GBFS menggunakan  $h(n)$ , dan A\* menggunakan gabungan  $g(n) + h(n)$ , di mana  $h(n)$  merupakan heuristik yang dipilih pengguna. Untuk IDS, pencarian dilakukan hingga kedalaman tertentu. Setelah solusi ditemukan, program akan melakukan `reconstructPath()` dan menampilkan langkah-langkah solusi melalui tampilan paginasi atau animasi.

4. Program menampilkan solusi permainan Rush Hour di GUI dan memberikan pilihan untuk menyimpan solusi ke dalam file teks (.txt).

Jika ditemukan solusi permainan, program akan menampilkan solusi permainan dalam bentuk susunan papan permainan yang setiap *piece*-nya disajikan dalam warna yang berbeda di GUI. Program akan menampilkan pesan seperti pada Gambar 6 apabila file masukan permainan tidak memiliki solusi. Selain itu, program akan menampilkan jumlah gerakan (*node*) yang telah dilakukan serta waktu eksekusi algoritma *pathfinding*. Pengguna dapat menyimpan langkah-langkah solusi ke dalam file teks sesuai lokasi yang diinginkan dengan menekan tombol *Save Solution*. Berikut merupakan contoh pada Gambar 7.



Gambar 6. Tampilan pesan apabila program tidak menemukan solusi



Gambar 7. Tampilan program setelah berhasil menemukan dan menyimpan solusi dalam file teks (.txt)

## BAB III IMPLEMENTASI ALGORITMA

### 3.1.Solver

Merupakan *interface* untuk kelas-kelas *Solver* spesifik tiap algoritma *pathfinding* yang berisi definisi *method* yang perlu diimplementasikan oleh mereka. Terdapat dalam file *Solver.java* yang memiliki *interface* *Solver*.

<i>Method</i>	<b>Deskripsi</b>
List<Board> solve(Board initialBoard)	Menyelesaikan <i>puzzle</i> Rush Hour dari papan awal dengan salah satu algoritma <i>pathfinding</i>
int getVisitedNodeCount()	Mengembalikan jumlah simpul yang dikunjungi dalam penyelesaian <i>puzzle</i>
long getExecutionTime()	Mengembalikan waktu eksekusi penyelesaian <i>puzzle</i>
private List<Board> reconstructPath(Board goal)	Mengembalikan <i>list</i> yang merupakan susunan jalur dari <i>initial node</i> hingga <i>goal node</i>

### 3.2.Algoritma Pathfinding GBFS (Greedy Best First Search)

Berisi *method* untuk mencari penyelesaian *puzzle* Rush Hour dengan algoritma pencarian jalur (*pathfinding*) *Greedy Best First Search*. Terdapat dalam file *GBFSSolver.java* yang memiliki kelas *GBFSSolver*. GBFS adalah algoritma pencarian jalur (*pathfinding*) yang mengandalkan fungsi heuristik  $h(n)$  sebagai estimasi jarak dari simpul saat ini ke tujuan, dan memilih node dengan nilai  $h(n)$  terkecil.

<i>Method</i>	<b>Deskripsi</b>
public GBFSSolver(String heuristicName)	Menyimpan jenis heuristik yang akan dipakai sebagai $h(n)$ dalam mengevaluasi simpul yang akan dikunjungi selanjutnya
private int heuristic(Board board)	Mengembalikan nilai $h(n)$ dari fungsi heuristik untuk evaluasi simpul

public List<Board> solve(Board start)	Menyelesaikan <i>puzzle</i> Rush Hour dari papan awal dengan algoritma <i>pathfinding</i> GBFS
private List<Board> reconstructPath(Board goal)	Mengembalikan <i>list</i> yang merupakan susunan jalur dari <i>initial node</i> hingga <i>goal node</i>
int getVisitedNodeCount()	Mengembalikan jumlah simpul yang dikunjungi dalam penyelesaian <i>puzzle</i>
long getExecutionTime()	Mengembalikan waktu eksekusi penyelesaian <i>puzzle</i>

### 3.3. Algoritma *Pathfinding UCS (Uniform Cost Search)*

Berisi *method* untuk mencari penyelesaian *puzzle* Rush Hour dengan algoritma pencarian jalur (*pathfinding*) *Uniform Cost Search* serta *inner class* Node untuk menyimpan informasi suatu simpul. Terdapat dalam file UCSSolver.java yang memiliki kelas UCSSolver.

<b>Method</b>	<b>Deskripsi</b>
public List<Board> solve(Board start)	Menyelesaikan <i>puzzle</i> Rush Hour dari papan awal dengan algoritma <i>pathfinding</i> UCS
private List<Board> reconstructPath(Board goal)	Mengembalikan <i>list</i> yang merupakan susunan jalur dari <i>initial node</i> hingga <i>goal node</i>
int getVisitedNodeCount()	Mengembalikan jumlah simpul yang dikunjungi dalam penyelesaian <i>puzzle</i>
long getExecutionTime()	Mengembalikan waktu eksekusi penyelesaian <i>puzzle</i>

### 3.4. Algoritma *Pathfinding A\* (A-Star)*

Berisi *method* untuk mencari penyelesaian *puzzle* Rush Hour dengan algoritma pencarian jalur (*pathfinding*) *A-Star* serta *inner class* Node untuk menyimpan informasi suatu simpul. Terdapat dalam file AStarSolver.java yang memiliki kelas AStarSolver. Algoritma A\* digunakan untuk menyelesaikan *puzzle* Rush Hour dengan menggabungkan biaya riil dari awal ke simpul saat ini dan estimasi jarak tersisa ke tujuan.

<b>Method</b>	<b>Deskripsi</b>
public AStarSolver(String heuristicName)	Menyimpan jenis heuristik yang akan dipakai sebagai $h(n)$ dalam mengevaluasi simpul yang akan dikunjungi selanjutnya
public List<Board> solve(Board start)	Menyelesaikan <i>puzzle</i> Rush Hour dari papan awal dengan algoritma <i>pathfinding</i> A-Star
private List<Board> reconstructPath(Board goal)	Mengembalikan <i>list</i> yang merupakan susunan jalur dari <i>initial node</i> hingga <i>goal node</i>
int getVisitedNodeCount()	Mengembalikan jumlah simpul yang dikunjungi dalam penyelesaian <i>puzzle</i>
long getExecutionTime()	Mengembalikan waktu eksekusi penyelesaian <i>puzzle</i>

### 3.5. Algoritma *Pathfinding IDS (Iterative Deepening Search)* (Bonus)

Berisi *method* untuk mencari penyelesaian *puzzle* Rush Hour dengan algoritma pencarian jalur (*pathfinding*) *Iterative Deepening Search*. Terdapat dalam file IDSSolver.java yang memiliki kelas IDSSolver. Algoritma pencarian *Iterative Deepening Search* (IDS) digunakan untuk menyelesaikan *puzzle* Rush Hour.

<b>Method</b>	<b>Deskripsi</b>
public List<Board> solve(Board start)	Menyelesaikan <i>puzzle</i> Rush Hour dari papan awal dengan algoritma <i>pathfinding</i> IDS
private List<Board> dfs(Board current, int depth, Set<Board> visited)	Melakukan penelusuran secara <i>Depth First Search</i> (DFS)
private List<Board> reconstructPath(Board goal)	Mengembalikan <i>list</i> yang merupakan susunan jalur dari <i>initial node</i> hingga <i>goal node</i>
int getVisitedNodeCount()	Mengembalikan jumlah simpul yang dikunjungi dalam penyelesaian <i>puzzle</i>

long getExecutionTime()	Mengembalikan waktu eksekusi penyelesaian <i>puzzle</i>
-------------------------	---

### 3.6. Jenis Heuristik (Bonus)

Berisi *method* dari tiga jenis heuristik yang dapat digunakan untuk pemilihan simpul yang akan ditelusuri selanjutnya. Terdapat dalam file Heuristic.java yang memiliki kelas Heuristic.

<i>Method</i>	<b>Deskripsi</b>
public static int blockingPiecesCount(Board board)	Heuristik ini menghitung jumlah kendaraan yang menghalangi jalan dari <i>primary piece</i> P ke pintu keluar K
public static int blockingPiecesWithMovability(Board board)	Heuristik ini menghitung jumlah kendaraan yang menghalangi jalan dari <i>primary piece</i> P ke pintu keluar K serta mempertimbangkan gerakan <i>piece</i> tersebut, apabila <i>piece</i> tersebut tidak bisa digerakan ke arah yang berlawanan dari orientasi <i>primary piece</i> , fungsi heuristik ditambahkan dua
public static int distanceToExit(Board board)	Heuristik ini menghitung jarak langsung, secara kolom atau baris, dari ujung depan <i>primary piece</i> ke pintu keluar
private static int[] findFrontOfP(Board board)	<i>Helper function</i> untuk menemukan ujung paling depan dari <i>piece</i>
private static boolean canMoveVertically(Board board, char vehicle)	Mengembalikan <i>true</i> jika <i>piece</i> dapat digerakan secara vertikal
private static boolean canMoveHorizontally(Board board, char vehicle)	Mengembalikan <i>true</i> jika <i>piece</i> dapat digerakan secara horizontal
public static int evaluate(Board board, String name)	Menjalankan heuristik sesuai pilihan jenisnya

### 3.7.Board

Berisi *method* untuk menyimpan serta memodifikasi papan permainan dalam Rush Hour. Terdapat dalam file Board.java yang memiliki kelas Board.

<i>Method</i>	<b>Deskripsi</b>
public Board(char[][] grid, int rows, int cols, int exitRow, int exitCol)	<i>Constructor</i> lengkap, menginisialisasi papan dengan ukuran dan posisi exit.
public void setExitPosition(int row, int col)	Menentukan koordinat pintu keluar dari papan.
public int getExitRow()	Mengembalikan baris posisi pintu keluar.
public int getExitCol()	Mengembalikan kolom posisi pintu keluar.
public char[][] getGrid()	Getter untuk grid papan.
public int getRows()	Getter untuk jumlah baris.
public int getCols()	Getter untuk jumlah kolom.
public boolean isGoal()	Mengecek apakah primary piece 'P' sudah bisa keluar melalui pintu exit.
public java.util.List<Board> getNeighbors()	Menghasilkan daftar board hasil pergerakan kendaraan satu per satu.
public boolean canExitDirectly()	Mengecek apakah primary piece 'P' bisa langsung keluar dari posisi sekarang.
public char[][] copyGridWithPRemoved()	Mengembalikan salinan grid dengan semua posisi 'P' dihapus.
public Board moveVehicleUntilBlocked(char vehicle, boolean horizontal, boolean negativeDirection)	Memindahkan kendaraan sampai tidak bisa bergerak lagi ke arah tertentu.
private boolean canMoveOneStep(char vehicle, boolean horizontal, boolean negativeDirection)	Mengecek apakah kendaraan bisa bergerak satu langkah ke arah tertentu.



private List<Board> getAllMovesForVehicle(char vehicle, boolean horizontal)	Mengembalikan semua pergerakan valid (dua arah) untuk kendaraan tertentu.
private Board moveVehicle(char vehicle, boolean horizontal, boolean negativeDirection)	Membuat board baru hasil pergerakan satu langkah kendaraan ke arah tertentu.
public String toString()	Menampilkan representasi papan sebagai string, termasuk posisi exit jika ada.
public String toStringWithExit()	Menampilkan representasi papan sebagai string, termasuk posisi exit jika ada.
public boolean equals(Object o)	Menentukan apakah dua board identik dari sisi isi grid-nya.
public int hashCode()	Menghasilkan hash berdasarkan isi grid, untuk keperluan koleksi seperti Set/Map.

### 3.8.Input Parser

Berisi *method* untuk membaca file konfigurasi permainan *puzzle* Rush Hour. Terdapat dalam file InputParser.java yang memiliki kelas InputParser.

<b>Method</b>	<b>Deskripsi</b>
public static Board parse(File file) throws FileNotFoundException	Membaca file konfigurasi papan permainan, memvalidasi format dan isi grid, serta mengembalikan objek Board dengan konfigurasi papan, posisi kendaraan, dan letak keluar K.
private static void validateGrid(char[][] grid)	Memvalidasi isi grid: memastikan tiap baris memiliki panjang yang konsisten, hanya berisi huruf kapital atau '.', memeriksa ukuran dan bentuk kendaraan (harus lurus dan kontigu), serta memastikan hanya ada satu kendaraan utama 'P'.

### 3.9.Main dengan GUI (Bonus)

Berisi *method* untuk menjalankan program “*Rush Hour Puzzle Solver*” secara keseluruhan yang ditampilkan di GUI. Terdapat dalam file GUIApp.java yang memiliki class GUIApp.

<i>Method</i>	<b>Deskripsi</b>
public void start(Stage primaryStage)	Method utama JavaFX yang memulai dan menampilkan antarmuka GUI aplikasi.
private void initializeColorPalette()	Menginisialisasi warna-warna untuk tiap kendaraan ( <i>piece</i> ) berdasarkan karakternya.
private void drawBoard(Board board)	Menampilkan papan permainan ke layar berdasarkan state dari objek Board.
private void displayStep(int step)	Menampilkan kondisi papan dan informasi pada langkah (step) tertentu dari solusi.
public static void main(String[] args)	Titik masuk program Java yang menjalankan aplikasi JavaFX.
public VBox getRightPanel()	Mengembalikan panel sisi kanan GUI yang berisi informasi tambahan seperti status.
public void setRightPanel(VBox rightPanel)	Mengatur ulang panel sisi kanan GUI dengan konten baru jika dibutuhkan.

## BAB IV

### SOURCE CODE PROGRAM

#### 4.1.Solver.java

```
J Solver.java x
1  package algo;
2
3  import java.util.List;
4
5  import model.Board;
6
7  public interface Solver {
8      // solves the puzzle and returns the solution path
9      List<Board> solve(Board initialBoard);
10
11      // returns the number of nodes visited during the search
12      int getVisitedNodeCount();
13
14      // returns the time taken to solve the puzzle in ms
15      long getExecutionTime();
16
17      // reconstructs the path from the initial board to the goal board
18      List<Board> reconstructPath(Board goal);
19  }
20
21
```

## 4.2.GBFSSolver.java

```
J GBFSSolver.java X
1  package algo;
2
3  import java.util.ArrayList;
4  import java.util.Comparator;
5  import java.util.HashSet;
6  import java.util.List;
7  import java.util.PriorityQueue;
8  import java.util.Set;
9
10 import model.Board;
11 import util.Heuristic;
12
13 public class GBFSSolver implements Solver {
14     private int visitedNodes = 0;
15     private long executionTime = 0;
16     private final String heuristicName;
17
18     public GBFSSolver(String heuristicName) {
19         this.heuristicName = heuristicName;
20     }
21
22     private int heuristic(Board board) {
23         return Heuristic.evaluate(board, heuristicName);
24     }
25
26     @Override
27     public List<Board> solve(Board start) {
28         long startTime = System.currentTimeMillis();
29
30         PriorityQueue<Board> pq = new PriorityQueue<>(Comparator.comparingInt(this::heuristic));
31         Set<Board> visited = new HashSet<>();
32
33         pq.add(start);
34
35         while (!pq.isEmpty()) {
36             Board current = pq.poll();
37             visitedNodes++;
38
39             if (current.isGoal()) {
40                 executionTime = System.currentTimeMillis() - startTime;
41                 return reconstructPath(current);
42             }
43
44             if (visited.contains(current)) continue;
45             visited.add(current);
46
47             for (Board neighbor : current.getNeighbors()) {
48                 neighbor.parent = current;
49                 pq.add(neighbor);
50             }
51         }
52
53         executionTime = System.currentTimeMillis() - startTime;
54         return new ArrayList<>();
55     }
56 }
```

```

57     @Override
58     public List<Board> reconstructPath(Board goal) {
59         List<Board> path = new ArrayList<>();
60         for (Board b = goal; b != null; b = b.parent) {
61             path.add(index:0, b);
62         }
63         return path;
64     }
65
66     @Override
67     public int getVisitedNodeCount() {
68         return visitedNodes;
69     }
70
71     @Override
72     public long getExecutionTime() {
73         return executionTime;
74     }
75 }
76

```

### 4.3.UCSSolver.java

```

UCSSolver.java x
1  package algo;
2
3  import java.util.ArrayList;
4  import java.util.Comparator;
5  import java.util.HashSet;
6  import java.util.List;
7  import java.util.PriorityQueue;
8  import java.util.Set;
9
10 import model.Board;
11
12 public class UCSolver implements Solver {
13     private int visitedNodeCount = 0;
14     private long executionTime = 0;
15
16     @Override
17     public List<Board> solve(Board start) {
18         long startTime = System.currentTimeMillis();
19
20         PriorityQueue<Node> pq = new PriorityQueue<>(Comparator.comparingInt(n -> n.cost));
21         Set<Board> visited = new HashSet<>();
22
23         pq.add(new Node(start, cost:0));
24
25         while (!pq.isEmpty()) {
26             Node current = pq.poll();
27             visitedNodeCount++;
28
29             if (current.board.isGoal()) {
30                 executionTime = System.currentTimeMillis() - startTime;
31                 return reconstructPath(current.board);
32             }
33         }
34     }
35 }

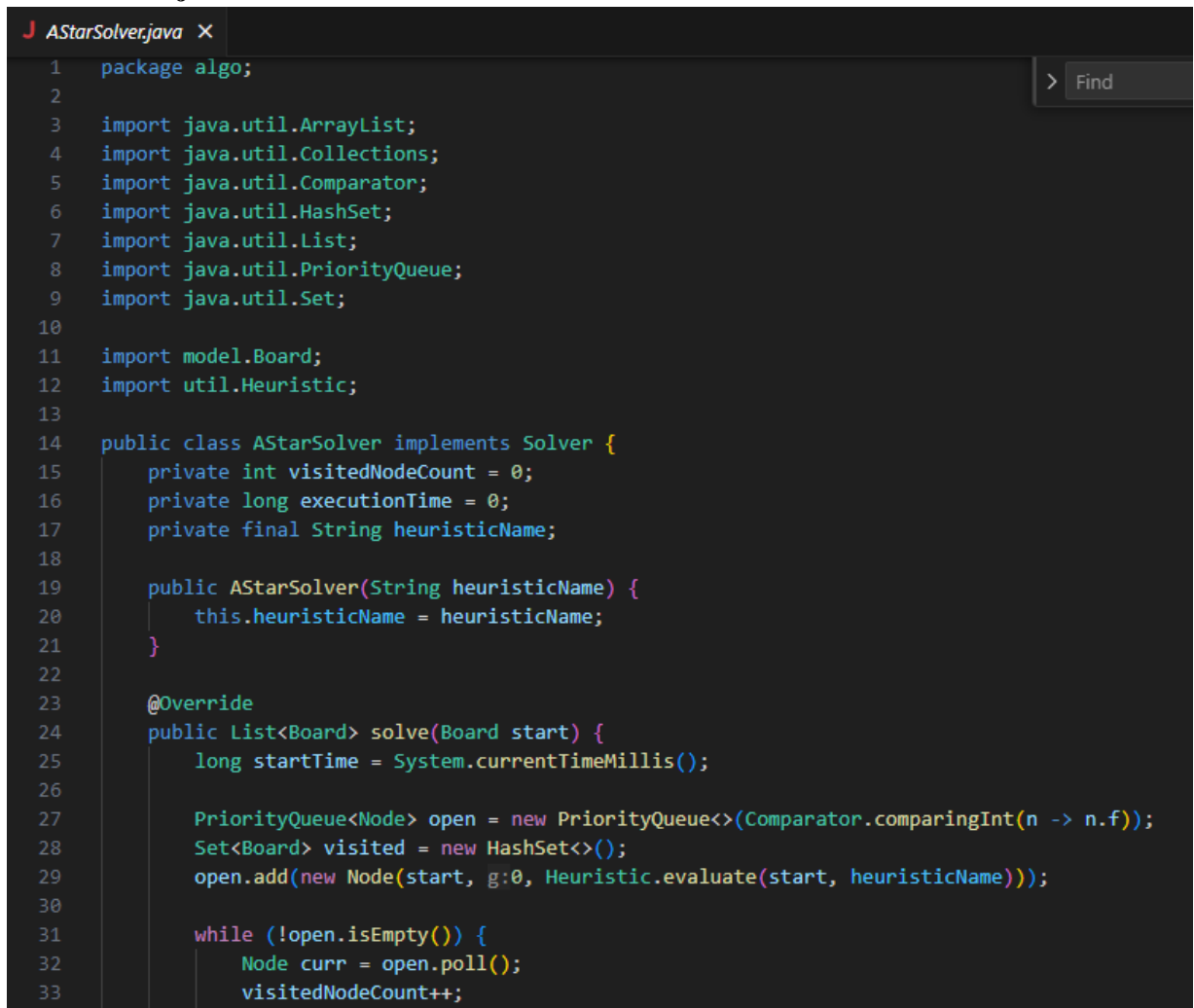
```

```

34         if (visited.contains(current.board)) continue;
35         visited.add(current.board);
36
37         for (Board neighbor : current.board.getNeighbors()) {
38             neighbor.parent = current.board;
39             pq.add(new Node(neighbor, current.cost + 1));
40         }
41     }
42
43     executionTime = System.currentTimeMillis() - startTime;
44     return new ArrayList<>();
45 }
46
47 @Override
48 public int getVisitedNodeCount() {
49     return visitedNodeCount;
50 }
51
52 @Override
53 public long getExecutionTime() {
54     return executionTime;
55 }
56
57 private static class Node {
58     Board board;
59     int cost;
60
61     Node(Board board, int cost) {
62         this.board = board;
63         this.cost = cost;
64     }
65 }
66
67 @Override
68 public List<Board> reconstructPath(Board goal) {
69     List<Board> path = new ArrayList<>();
70     for (Board b = goal; b != null; b = b.parent) {
71         path.add(index:0, b);
72     }
73     return path;
74 }
75 }
76

```

#### 4.4.AStarSolver.java



```
1 package algo;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.Comparator;
6 import java.util.HashSet;
7 import java.util.List;
8 import java.util.PriorityQueue;
9 import java.util.Set;
10
11 import model.Board;
12 import util.Heuristic;
13
14 public class AStarSolver implements Solver {
15     private int visitedNodeCount = 0;
16     private long executionTime = 0;
17     private final String heuristicName;
18
19     public AStarSolver(String heuristicName) {
20         this.heuristicName = heuristicName;
21     }
22
23     @Override
24     public List<Board> solve(Board start) {
25         long startTime = System.currentTimeMillis();
26
27         PriorityQueue<Node> open = new PriorityQueue<>(Comparator.comparingInt(n -> n.f));
28         Set<Board> visited = new HashSet<>();
29         open.add(new Node(start, g:0, Heuristic.evaluate(start, heuristicName)));
30
31         while (!open.isEmpty()) {
32             Node curr = open.poll();
33             visitedNodeCount++;
```

```

35         if (curr.board.isGoal()) {
36             executionTime = System.currentTimeMillis() - startTime;
37             return reconstructPath(curr.board);
38         }
39
40         visited.add(curr.board);
41         for (Board neighbor : curr.board.getNeighbors()) {
42             if (!visited.contains(neighbor)) {
43                 int g = curr.g + 1;
44                 int h = Heuristic.evaluate(neighbor, heuristicName);
45                 open.add(new Node(neighbor, g, g + h));
46                 neighbor.parent = curr.board;
47             }
48         }
49     }
50
51     executionTime = System.currentTimeMillis() - startTime;
52     return new ArrayList<>(); // tidak ada solusi
53 }
54
55 @Override
56 public List<Board> reconstructPath(Board goal) {
57     List<Board> path = new ArrayList<>();
58     for (Board at = goal; at != null; at = at.parent)
59         path.add(at);
60     Collections.reverse(path);
61     return path;
62 }

```

```

64     @Override
65     public int getVisitedNodeCount() {
66         return visitedNodeCount;
67     }
68
69     @Override
70     public long getExecutionTime() {
71         return executionTime;
72     }
73
74     private static class Node {
75         Board board;
76         int g, f;
77
78         Node(Board b, int g, int f) {
79             this.board = b;
80             this.g = g;
81             this.f = f;
82         }
83     }
84 }

```



#### 4.5.IDSSolver.java

```
J IDSSolver.java X
1  package algo;
2
3  import java.util.ArrayList;
4  import java.util.HashSet;
5  import java.util.List;
6  import java.util.Set;
7
8  import model.Board;
9
10 public class IDSSolver implements Solver {
11     private final int maxDepth;
12     private int visitedNodeCount = 0;
13     private long executionTime = 0;
14
15     public IDSSolver(int maxDepth) {
16         this.maxDepth = maxDepth;
17     }
18
19     @Override
20     public List<Board> solve(Board start) {
21         long startTime = System.currentTimeMillis();
22
23         for (int depth = 0; depth <= maxDepth; depth++) {
24             Set<Board> visited = new HashSet<>();
25             List<Board> result = dfs(start, depth, visited);
26
27             if (!result.isEmpty()) {
28                 executionTime = System.currentTimeMillis() - startTime;
29                 return result;
30             }
31         }
32     }
33 }
```

```

33     executionTime = System.currentTimeMillis() - startTime;
34     return new ArrayList<>();
35 }
36
37 private List<Board> dfs(Board current, int depth, Set<Board> visited) {
38     visitedNodeCount++;
39     if (depth < 0) return new ArrayList<>();
40     if (current.isGoal()) return reconstructPath(current);
41
42     visited.add(current);
43
44     for (Board neighbor : current.getNeighbors()) {
45         if (!visited.contains(neighbor)) {
46             neighbor.parent = current;
47             List<Board> result = dfs(neighbor, depth - 1, visited);
48             if (!result.isEmpty()) return result;
49         }
50     }
51
52     return new ArrayList<>();
53 }
54
55 @Override
56 public List<Board> reconstructPath(Board goal) {
57     List<Board> path = new ArrayList<>();
58     for (Board b = goal; b != null; b = b.parent) {
59         path.add(index:0, b);
60     }
61     return path;
62 }
63
64 @Override
65 public int getVisitedNodeCount() {
66     return visitedNodeCount;
67 }
68
69 @Override
70 public long getExecutionTime() {
71     return executionTime;
72 }
73 }

```

## 4.6.Heuristic.java

```
J Heuristic.java ×
1  package util;
2
3  import java.util.ArrayList;
4  import java.util.HashSet;
5  import java.util.List;
6  import java.util.Set;
7
8  import model.Board;
9
10 public class Heuristic {
11
12     // First Heuristic: Blocking Pieces Count
13     public static int blockingPiecesCount(Board board) {
14         // Cari posisi 'P'
15         int pRow = -1, pCol = -1;
16         outer:
17         for (int i = 0; i < board.rows; i++) {
18             for (int j = 0; j < board.cols; j++) {
19                 if (board.grid[i][j] == 'P') {
20                     pRow = i;
21                     pCol = j;
22                     break outer;
23                 }
24             }
25         }
26
27         // Cari orientasi 'P'
28         boolean isHorizontal = (pCol + 1 < board.cols && board.grid[pRow][pCol + 1] == 'P');
29
30         // Cari posisi 'K' (exit)
31         int exitRow = board.getExitRow();
32         int exitCol = board.getExitCol();
```

```

34     int count = 0;
35
36     if (isHorizontal) {
37         // Bergerak horizontal ke arah exit
38         int dir = Integer.compare(exitCol, pCol); // +1 atau -1
39         int j = pCol;
40         while ((j += dir) >= 0 && j < board.cols) {
41             char c = board.grid[pRow][j];
42             if (c != '.' && c != 'K' && c != 'P') count++;
43         }
44     } else {
45         // Bergerak vertikal ke arah exit
46         int dir = Integer.compare(exitRow, pRow);
47         int i = pRow;
48         while ((i += dir) >= 0 && i < board.rows) {
49             char c = board.grid[i][pCol];
50             if (c != '.' && c != 'K' && c != 'P') count++;
51         }
52     }
53
54     return count;
55 }
56
57 // Second Heuristic: Blocking Pieces with Movability Penalty
58 public static int blockingPiecesWithMovability(Board board) {
59     int[] pRowCol = findFrontOfP(board);
60     if (pRowCol == null) return Integer.MAX_VALUE;
61     int row = pRowCol[0];
62     int col = pRowCol[1];
63
64     int exitCol = board.getExitCol();
65     int score = 0;
66     Set<Character> seen = new HashSet<>();
67
68     boolean isPrimaryHorizontal = isPrimaryPieceHorizontal(board);
69
70     for (int j = col + 1; j < board.getCols() && j <= exitCol; j++) {
71         char block = board.grid[row][j];
72         if (block != '.' && block != 'K' && block != 'P' && !seen.contains(block)) {
73             seen.add(block);
74             score += 1;
75
76             if (isPrimaryHorizontal) {
77                 if (!canMoveVertically(board, block)) {
78                     score += 2;
79                 }
80             } else {
81                 if (!canMoveHorizontally(board, block)) {
82                     score += 2;
83                 }
84             }
85         }
86     }
87
88     return score;
89 }

```

```

91  private static boolean canMoveVertically(Board board, char vehicle) {
92      for (int i = 0; i < board.getRows(); i++) {
93          for (int j = 0; j < board.getCols(); j++) {
94              if (board.grid[i][j] == vehicle) {
95                  boolean up = i > 0 && board.grid[i - 1][j] == '.';
96                  boolean down = i < board.getRows() - 1 && board.grid[i + 1][j] == '.';
97                  return up || down;
98              }
99          }
100      }
101      return false;
102  }
103
104  private static boolean canMoveHorizontally(Board board, char vehicle) {
105      for (int i = 0; i < board.getRows(); i++) {
106          for (int j = 0; j < board.getCols(); j++) {
107              if (board.grid[i][j] == vehicle) {
108                  boolean left = j > 0 && board.grid[i][j - 1] == '.';
109                  boolean right = j < board.getCols() - 1 && board.grid[i][j + 1] == '.';
110                  return left || right;
111              }
112          }
113      }
114      return false;
115  }
116
117  // Third Heuristic: Distance to Exit (Horizontal or Vertical)
118  public static int distanceToExit(Board board) {
119      int[] pHead = findFrontOfP(board);
120      if (pHead == null) return Integer.MAX_VALUE;
121      int pRow = pHead[0];
122      int pCol = pHead[1];
123
124      int exitRow = board.getExitRow();
125      int exitCol = board.getExitCol();
126      if (exitRow < 0 || exitCol < 0) return Integer.MAX_VALUE;
127
128      // Tentukan orientasi P
129      boolean isHorizontal = false;
130      if (pCol + 1 < board.cols && board.grid[pRow][pCol + 1] == 'P') {
131          isHorizontal = true;
132      }
133
134      if (isHorizontal) {
135          // Bergerak horizontal
136          if (pRow != exitRow) return Integer.MAX_VALUE; // bukan arah keluar
137          return Math.abs(exitCol - pCol);
138      } else {
139          // Bergerak vertikal
140          if (pCol != exitCol) return Integer.MAX_VALUE; // bukan arah keluar
141          return Math.abs(exitRow - pRow);
142      }
143  }

```

```

163 // Helper: Find front-right most position of P
164 private static int[] findFrontOfP(Board board) {
165     int lastCol = -1;
166     int pRow = -1;
167     for (int i = 0; i < board.getRows(); i++) {
168         for (int j = 0; j < board.getCols(); j++) {
169             if (board.grid[i][j] == 'P') {
170                 if (j > lastCol) {
171                     lastCol = j;
172                     pRow = i;
173                 }
174             }
175         }
176     }
177     return (lastCol != -1) ? new int[]{pRow, lastCol} : null;
178 }
179
180 // Dispatcher
181 public static int evaluate(Board board, String name) {
182     return switch (name) {
183         case "Blocking Pieces Count" -> blockingPiecesCount(board);
184         case "Blocking Pieces With Movability" -> blockingPiecesWithMovability(board);
185         case "Distance-to-Exit" -> distanceToExit(board);
186         default -> blockingPiecesCount(board);
187     };
188 }
189 }

```

## 4.7.Board.java

```
J Board.java X
1  package model;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.Comparator;
6  import java.util.List;
7
8  public class Board {
9      public char[][] grid;
10     public int rows, cols;
11     public String move; // Deskripsi gerakan terakhir (opsional)
12     public Board parent; // Parent untuk tracking path (opsional)
13
14     // Posisi pintu keluar, bisa di luar grid (misal exitCol == cols artinya di luar sebelah kanan)
15     private int exitRow = -1;
16     private int exitCol = -1;
17
18     // Constructor lengkap
19     public Board(char[][] grid, int rows, int cols, int exitRow, int exitCol) {
20         this.rows = rows;
21         this.cols = cols;
22         this.grid = new char[rows][cols];
23         for (int i = 0; i < rows; i++) {
24             this.grid[i] = Arrays.copyOf(grid[i], cols);
25         }
26         this.exitRow = exitRow;
27         this.exitCol = exitCol;
28     }
29
30     // Constructor tanpa exit (default exit pos = -1)
31     public Board(char[][] grid) {
32         this.rows = grid.length;
33         this.cols = (rows > 0) ? grid[0].length : 0;
34         this.grid = new char[rows][cols];
35         for (int i = 0; i < rows; i++) {
36             this.grid[i] = Arrays.copyOf(grid[i], cols);
37         }
38     }
39
40     public void setExitPosition(int row, int col) {
41         this.exitRow = row;
42         this.exitCol = col;
43     }
44
45     public int getExitRow() { return exitRow; }
46     public int getExitCol() { return exitCol; }
47     public char[][] getGrid() { return grid; }
48     public int getRows() { return rows; }
49     public int getCols() { return cols; }
```

```

51 // Check if the board is in goal state: primary piece 'P' exits through 'K'
52 public boolean isGoal() {
53     java.util.List<int[]> pPositions = new java.util.ArrayList<>();
54     for (int i = 0; i < rows; i++) {
55         for (int j = 0; j < cols; j++) {
56             if (grid[i][j] == 'P') {
57                 pPositions.add(new int[]{i, j});
58             }
59         }
60     }
61
62     if (pPositions.isEmpty()) return false;
63
64     boolean isHorizontal = pPositions.stream().allMatch(p -> p[0] == pPositions.get(index:0)[0]);
65     boolean isVertical = pPositions.stream().allMatch(p -> p[1] == pPositions.get(index:0)[1]);
66
67     if (!isHorizontal && !isVertical) return false;
68
69     // urutkan dari depan ke belakang
70     pPositions.sort((a, b) -> isHorizontal ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));
71
72     if (isHorizontal) {
73         int row = pPositions.get(index:0)[0];
74         int leftMost = pPositions.get(index:0)[1];
75         int rightMost = pPositions.get(pPositions.size() - 1)[1];
76
77         // Exit di kanan
78         if (exitRow == row && exitCol == rightMost + 1) {
79             for (int c = rightMost + 1; c < cols; c++) {
80                 if (grid[row][c] != '.') return false;
81             }
82             return true;
83         }
84
85         // Exit di kiri
86         if (exitRow == row && exitCol == leftMost - 1) {
87             for (int c = leftMost - 1; c >= 0; c--) {
88                 if (grid[row][c] != '.') return false;
89             }
90             return true;
91         }
92
93     } else if (isVertical) {
94         int col = pPositions.get(index:0)[1];
95         int topMost = pPositions.get(index:0)[0];
96         int bottomMost = pPositions.get(pPositions.size() - 1)[0];
97
98         // Exit di bawah
99         if (exitCol == col && exitRow == bottomMost + 1) {
100             for (int r = bottomMost + 1; r < rows; r++) {
101                 if (grid[r][col] != '.') return false;
102             }
103             return true;
104         }

```



```

106         // Exit di atas
107         if (exitCol == col && exitRow == topMost - 1) {
108             for (int r = topMost - 1; r >= 0; r--) {
109                 if (grid[r][col] != '.') return false;
110             }
111             return true;
112         }
113     }
114
115     return false;
116 }
117
118 public java.util.List<Board> getNeighbors() {
119     java.util.List<Board> neighbors = new java.util.ArrayList<>();
120
121     java.util.Set<Character> vehicles = new java.util.HashSet<>();
122     for (int i = 0; i < rows; i++) {
123         for (int j = 0; j < cols; j++) {
124             char c = grid[i][j];
125             if (c != '.' && c != 'K') {
126                 vehicles.add(c);
127             }
128         }
129     }

```

```

131     for (char v : vehicles) {
132         List<int[]> positions = new ArrayList<>();
133         for (int i = 0; i < rows; i++) {
134             for (int j = 0; j < cols; j++) {
135                 if (grid[i][j] == v) {
136                     positions.add(new int[]{i, j});
137                 }
138             }
139         }
140
141         if (positions.isEmpty()) continue;
142
143         boolean horizontal = true;
144         int firstRow = positions.get(index:0)[0];
145         for (int[] pos : positions) {
146             if (pos[0] != firstRow) {
147                 horizontal = false;
148                 break;
149             }
150         }
151
152         neighbors.addAll(getAllMovesForVehicle(v, horizontal));
153     }
154
155     // Tambahan: Jika primary piece P bisa langsung keluar (goal state), buat Board baru tanpa P
156     if (isGoal()) {
157         char[][] newGrid = new char[rows][cols];
158         for (int i = 0; i < rows; i++) {
159             newGrid[i] = Arrays.copyOf(grid[i], cols);
160         }

```

```

162         for (int i = 0; i < rows; i++) {
163             for (int j = 0; j < cols; j++) {
164                 if (newGrid[i][j] == 'P') {
165                     newGrid[i][j] = '.';
166                 }
167             }
168         }
169
170         if (canExitDirectly()) {
171             Board exitBoard = new Board(copyGridWithPRemoved(), rows, cols, exitRow, exitCol);
172             exitBoard.parent = this;
173             exitBoard.move = "Primary piece P exits through K";
174             neighbors.add(exitBoard);
175         }
176     }
177
178     return neighbors;
179 }
180
181 public boolean canExitDirectly() {
182     List<int[]> positions = new ArrayList<>();
183     for (int i = 0; i < rows; i++) {
184         for (int j = 0; j < cols; j++) {
185             if (grid[i][j] == 'P') {
186                 positions.add(new int[]{i, j});
187             }
188         }
189     }
190
191     if (positions.isEmpty()) return false;
192
193     positions.sort(Comparator.comparingInt(p -> p[0])); // sort by row
194     int topRow = positions.get(index:0)[0];
195     int bottomRow = positions.get(positions.size() - 1)[0];
196     int col = positions.get(index:0)[1];
197
198     positions.sort(Comparator.comparingInt(p -> p[1])); // sort by col
199     int leftCol = positions.get(index:0)[1];
200     int rightCol = positions.get(positions.size() - 1)[1];
201     int row = positions.get(index:0)[0];
202
203     // === Horizontal Exit ===
204     if (exitRow == row) {
205         // Exit to right
206         if (exitCol > rightCol) {
207             for (int j = rightCol + 1; j < exitCol; j++) {
208                 if (grid[row][j] != '.') return false;
209             }
210             return true;
211         }
212         // Exit to left
213         if (exitCol < leftCol) {
214             for (int j = exitCol + 1; j < leftCol; j++) {
215                 if (grid[row][j] != '.') return false;
216             }
217             return true;
218         }
219     }

```

```

221 // === Vertical Exit ===
222 if (exitCol == col) {
223     // Exit to bottom
224     if (exitRow > bottomRow) {
225         for (int i = bottomRow + 1; i < exitRow; i++) {
226             if (grid[i][col] != '.') return false;
227         }
228         return true;
229     }
230     // Exit to top
231     if (exitRow < topRow) {
232         for (int i = exitRow + 1; i < topRow; i++) {
233             if (grid[i][col] != '.') return false;
234         }
235         return true;
236     }
237 }
238
239 return false;
240 }
241
242 public char[][] copyGridWithPRemoved() {
243     char[][] newGrid = new char[rows][cols];
244     for (int i = 0; i < rows; i++) {
245         newGrid[i] = Arrays.copyOf(grid[i], cols);
246         for (int j = 0; j < cols; j++) {
247             if (newGrid[i][j] == 'P') newGrid[i][j] = '.';
248         }
249     }
250     return newGrid;
251 }

```

```

253 public Board moveVehicleUntilBlocked(char vehicle, boolean horizontal, boolean negativeDirection) {
254     Board current = this;
255     while (true) {
256         if (!current.canMoveOneStep(vehicle, horizontal, negativeDirection)) break;
257         Board next = current.moveVehicle(vehicle, horizontal, negativeDirection);
258         if (next == null) break;
259         current = next;
260     }
261     return current == this ? null : current;
262 }
263
264 private boolean canMoveOneStep(char vehicle, boolean horizontal, boolean negativeDirection) {
265     List<int[]> positions = new ArrayList<>();
266     for (int i = 0; i < rows; i++) {
267         for (int j = 0; j < cols; j++) {
268             if (grid[i][j] == vehicle) {
269                 positions.add(new int[]{i, j});
270             }
271         }
272     }
273
274     if (positions.isEmpty()) return false;
275
276     positions.sort((a, b) -> horizontal ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));
277
278     int dRow = horizontal ? 0 : (negativeDirection ? -1 : 1);
279     int dCol = horizontal ? (negativeDirection ? -1 : 1) : 0;
280
281     int[] edge = negativeDirection ? positions.get(index:0) : positions.get(positions.size() - 1);
282     int newRow = edge[0] + dRow;
283     int newCol = edge[1] + dCol;
284
285     if (newRow < 0 || newRow >= rows || newCol < 0 || newCol >= cols) return false;
286
287     // Cek apakah cell target kosong
288     return grid[newRow][newCol] == '.';
289 }
290
291 private List<Board> getAllMovesForVehicle(char vehicle, boolean horizontal) {
292     List<Board> neighbors = new ArrayList<>();
293
294     // Arah negatif dan positif (kiri-kanan atau atas-bawah)
295     for (boolean negative : new boolean[]{true, false}) {
296         Board moved = moveVehicleUntilBlocked(vehicle, horizontal, negative);
297         if (moved != null) {
298             neighbors.add(moved);
299         }
300     }
301
302     return neighbors;
303 }
304
305 private Board moveVehicle(char vehicle, boolean horizontal, boolean negativeDirection) {
306     char[][] newGrid = new char[rows][cols];
307     for (int i = 0; i < rows; i++) {
308         newGrid[i] = Arrays.copyOf(grid[i], cols);
309     }

```

```

311     for (int i = 0; i < rows; i++) {
312         for (int j = 0; j < cols; j++) {
313             if (newGrid[i][j] == vehicle) {
314                 newGrid[i][j] = '.';
315             }
316         }
317     }
318
319     java.util.List<int[]> positions = new java.util.ArrayList<>();
320     for (int i = 0; i < rows; i++) {
321         for (int j = 0; j < cols; j++) {
322             if (grid[i][j] == vehicle) {
323                 positions.add(new int[]{i, j});
324             }
325         }
326     }
327     positions.sort((a, b) -> horizontal ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));
328
329     int shiftRow = 0, shiftCol = 0;
330     if (horizontal) {
331         shiftCol = negativeDirection ? -1 : 1;
332     } else {
333         shiftRow = negativeDirection ? -1 : 1;
334     }
335
336     for (int[] pos : positions) {
337         int r = pos[0] + shiftRow;
338         int c = pos[1] + shiftCol;
339
340         if (r < 0 || r >= rows || c < 0 || c >= cols) {
341             if (vehicle == 'K') continue;
342             else return null;
343         }
344
345         newGrid[r][c] = vehicle;
346     }
347
348     Board newBoard = new Board(newGrid, rows, cols, exitRow, exitCol);
349     newBoard.parent = this;
350     newBoard.move = "Move " + vehicle + " " + (horizontal ? (negativeDirection ? "left" : "right") :
351         (negativeDirection ? "up" : "down"));
352     return newBoard;
353 }
354
355 @Override
356 public String toString() {
357     return toStringWithExit();
358 }
359
360 public String toStringWithExit() {
361     StringBuilder sb = new StringBuilder();
362
363     boolean pExited = isGoal();
364
365     if (exitRow == -1 && exitCol >= 0 && exitCol < cols) {
366         for (int j = 0; j < cols; j++) {
367             sb.append(j == exitCol ? 'K' : '.');
368         }
369         sb.append(c: '\n');
370     }

```

```

371         for (int i = 0; i < rows; i++) {
372             if (i == exitRow && exitCol == -1) sb.append(c: 'K');
373             for (int j = 0; j < cols; j++) {
374                 char c = grid[i][j];
375                 if (c == 'P' && pExited) {
376                     sb.append(c: '.');
377                 } else {
378                     sb.append(c);
379                 }
380             }
381             if (i == exitRow && exitCol == cols) sb.append(c: 'K');
382             sb.append(c: '\n');
383         }
384
385         if (exitRow == rows && exitCol >= 0 && exitCol < cols) {
386             for (int j = 0; j < cols; j++) {
387                 sb.append(j == exitCol ? 'K' : '.');
388             }
389             sb.append(c: '\n');
390         }
391
392         return sb.toString();
393     }
}

395 @Override
396 public boolean equals(Object o) {
397     if (!(o instanceof Board)) return false;
398     Board other = (Board) o;
399     return Arrays.deepEquals(this.grid, other.grid);
400 }
401
402 @Override
403 public int hashCode() {
404     return Arrays.deepHashCode(grid);
405 }
406 }

```

#### **4.8.InputParser.java**

```
InputParser.java 1 X
src > util > InputParser.java > ...
1  package util;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.util.ArrayList;
6  import java.util.HashSet;
7  import java.util.List;
8  import java.util.Scanner;
9  import java.util.Set;
10 import model.Board;
11
12 // fails to parse input file w K on top/bottom properly
13 public class InputParser {
14
15     public static Board parse(File file) throws FileNotFoundException {
16         Scanner scanner = new Scanner(file);
17         int rows = 0, cols = 0;
18         int pieceAmt = 0;
19         int kCount = 0;
20
21         // Baca baris konfigurasi ukuran papan
22         while (scanner.hasNextLine()) {
23             String line = scanner.nextLine().trim();
24             if (!line.isEmpty()) {
25                 String[] parts = line.split(regex: "\\s+");
26                 if (parts.length == 2) {
27                     rows = Integer.parseInt(parts[0]);
28                     cols = Integer.parseInt(parts[1]);
29                     break;
30                 }
31             }
32         }
33
34         while (scanner.hasNextLine()) {
35             String line = scanner.nextLine().trim();
36             if (!line.isEmpty()) {
37                 String[] parts = line.split(regex: "\\s+");
38                 if (parts.length == 1) {
39                     pieceAmt = Integer.parseInt(parts[0]);
40                     break;
41                 }
42             }
43         }
44
45         // Nilai A, B, dan N harus > 0
46         if (rows <= 0 || cols <= 0 || pieceAmt <= 0) {
47             throw new IllegalArgumentException(
48                 "A, B, and N must be greater than 0 (A=" + rows +
49                 ", B=" + cols + ", N=" + pieceAmt + ")");
50         }
51
52         List<String> rawLines = new ArrayList<>();
```



```

53
54     while (scanner.hasNextLine()) {
55         String line = scanner.nextLine().stripTrailing();
56         if (!line.isEmpty()) {
57             rawLines.add(line);
58         }
59     }
60
61     int exitRow = -1, exitCol = -1;
62     if (!rawLines.isEmpty()) {
63         String top = rawLines.get(index:0);
64         if (top.trim().equals(anObject:"K")) {
65             kCount++;
66             exitRow = -1;
67             exitCol = top.length() - 1;
68             rawLines.remove(index:0);
69             if (exitCol > cols - 1 || exitCol <= 0) {
70                 throw new IllegalArgumentException(s:"The position of 'K' is invalid.");
71             }
72         }
73     }
74
75     if (!rawLines.isEmpty()) {
76         String bottom = rawLines.get(rawLines.size() - 1);
77         if (bottom.trim().equals(anObject:"K")) {
78             kCount++;
79             exitRow = rows;
80             exitCol = bottom.length() - 1;
81             rawLines.remove(rawLines.size() - 1);
82             if (exitCol > cols - 1 || exitCol <= 0) {
83                 throw new IllegalArgumentException(s:"The position of 'K' is invalid.");
84             }
85         }
86     }
87
88     if (!rawLines.isEmpty() && rawLines.get(index:0).length() >= cols) {
89         String top = rawLines.get(index:0);
90         boolean foundK = false;
91         for (int j = 0; j < cols; j++) {
92             if (top.charAt(j) == 'K') {
93                 if (!(j == 0 && top.length() > cols) || (j == top.length() - 1 && top.length() > cols)) {
94                     exitRow = j;
95                     exitCol = -1; // K di pinggir kiri
96                     kCount++;
97                     rawLines.remove(index:0);
98                     foundK = true;
99                     break;
100                 }
101             }
102         }
103         if (foundK && rawLines.size() < rows) {
104             throw new IllegalArgumentException(s:"Number of board rows is less than configuration after removing top K row.");
105         }

```

```

107
108     if (!rawLines.isEmpty() && rawLines.get(rawLines.size() - 1).length() >= cols) {
109         String bottom = rawLines.get(rawLines.size() - 1);
110         boolean foundK = false;
111         for (int j = 0; j < cols; j++) {
112             if (bottom.charAt(j) == 'K') {
113                 if (!(j == 0 && bottom.length() > cols) || (j == bottom.length() - 1 && bottom.length() > cols)) {
114                     exitRow = rows;
115                     exitCol = j;
116                     rawLines.remove(rawLines.size() - 1);
117                     foundK = true;
118                     break;
119                 }
120             }
121         }
122         if (foundK && rawLines.size() < rows) {
123             throw new IllegalArgumentException(s+"Number of board rows is less than configuration after removing bottom K row.");
124         }
125     }
126
127     if (rawLines.size() != rows) {
128         throw new IllegalArgumentException("Number of board rows (" + rawLines.size() + ") does not match the configured rows = " + rows);
129     }
130
131     for (int i = 0; i < rows; i++) {
132         String line = rawLines.get(i);
133         rawLines.set(i, line.trim());
134     }
135
136     for (int i = 0; i < rows; i++) {
137         String line = rawLines.get(i);
138
139         if (line.length() < cols) {
140             throw new IllegalArgumentException("Row " + i + " has length less than " + cols);
141         }
142
143         if (line.length() > cols) {
144             if (line.charAt(index:0) == 'K') {
145                 kCount++;
146                 exitRow = i;
147                 exitCol = -1; // K di pinggir kiri
148                 line = line.substring(beginIndex:1);
149             } else if (line.charAt(line.length() - 1) == 'K') {
150                 kCount++;
151                 exitRow = i;
152                 exitCol = cols; // K di pinggir kanan
153                 line = line.substring(beginIndex:0, line.length() - 1);
154             } else {
155                 throw new IllegalArgumentException("Row " + i + " has length more than " + cols + " without 'K' on the edge.");
156             }
157         }
158     }

```

```

157         if (line.length() != cols) {
158             throw new IllegalArgumentException("After trimming, row " + i + " length does not match the specified column count.");
159         }
160     }
161     rawLines.set(i, line);
162 }
163
164 for (int j = 0; j < cols; j++) {
165     if (line.charAt(j) == 'K') {
166         throw new IllegalArgumentException("Character 'K' must only appear at the edge of the board (row " + i + ", column " + j + ").");
167     }
168 }
169 }
170
171 char[][] grid = new char[rows][cols];
172 for (int i = 0; i < rows; i++) {
173     String line = rawLines.get(i);
174     for (int j = 0; j < cols; j++) {
175         char c = line.charAt(j);
176         if (c == 'K') {
177             exitRow = i;
178             exitCol = j;
179             grid[i][j] = '.'; // treat as empty
180         } else {
181             grid[i][j] = c;
182         }
183     }
184 }
185
186 validateGrid(grid);
187
188 // Validasi jumlah kendaraan based on N (tanpa P & K)
189 Set<Character> pieces = new HashSet<>();
190 for (int i = 0; i < rows; i++) {
191     for (int j = 0; j < cols; j++) {
192         char c = grid[i][j];
193         if (c != '.' && c != 'P') { // abaikan P dan titik
194             pieces.add(c);
195         }
196     }
197 }
198 if (pieces.size() != pieceAmt) {
199     throw new IllegalArgumentException(
200         "Number of different vehicles (" + pieces.size() +
201         ") does not match N = " + pieceAmt);
202 }
203
204 if (kCount != 1) {
205     throw new IllegalArgumentException(
206         "There must be exactly one 'K' character. Found " + kCount + ".");
207 }
208

```

```

209     if (exitRow == -1 && exitCol == -1) {
210         throw new IllegalArgumentException("Exit position 'K' not found or invalid.");
211     }
212     return new Board(grid, rows, cols, exitRow, exitCol);
213 }
214
215 private static void validateGrid(char[][] grid) {
216     int rows = grid.length;
217     int cols = grid[0].length;
218
219     Set<Character> visited = new HashSet<>();
220     Set<Character> usedChars = new HashSet<>();
221
222     boolean foundP = false;
223
224     for (int i = 0; i < rows; i++) {
225         if (grid[i].length != cols) {
226             throw new IllegalArgumentException("Row " + i + " has length " + grid[i].length + ", expected " + cols + ".");
227         }
228     }
229
230     for (int i = 0; i < rows; i++) {
231         for (int j = 0; j < cols; j++) {
232             char c = grid[i][j];
233
234             if (c != '.' && !Character.isUpperCase(c)) {
235                 throw new IllegalArgumentException("Grid may only contain uppercase letters and '.' (invalid character '" + c + "' at row " + i + ", column " + j + ".");
236             }
237
238             if (c == '.' || visited.contains(c)) continue;
239
240             visited.add(c);
241             List<int[]> positions = new ArrayList<>();
242             positions.add(new int[]{i, j});
243
244             for (int x = 0; x < rows; x++) {
245                 for (int y = 0; y < cols; y++) {
246                     if ((x != i || y != j) && grid[x][y] == c) {
247                         positions.add(new int[]{x, y});
248                     }
249                 }
250             }
251
252             if (usedChars.contains(c)) {
253                 throw new IllegalArgumentException("Character '" + c + "' is used by more than one vehicle.");
254             }
255             usedChars.add(c);
256
257             if (positions.size() < 2 || positions.size() > 3) {
258                 throw new IllegalArgumentException("Vehicle '" + c + "' has size " + positions.size() + " cells. Vehicle length must be 2 or 3 cells.");
259             }
260         }
261     }
262 }

```

```

260
261     boolean sameRow = true, sameCol = true;
262     int baseRow = positions.get(index:0)[0];
263     int baseCol = positions.get(index:0)[1];
264     for (int[] pos : positions) {
265         if (pos[0] != baseRow) sameRow = false;
266         if (pos[1] != baseCol) sameCol = false;
267     }
268     if (!sameRow && !sameCol) {
269         throw new IllegalArgumentException("Vehicle '" + c + "' is not aligned straight (possibly diagonal).");
270     }
271
272     // setelah pengecekan sameRow / sameCol:
273     if (sameRow) {
274         int min = positions.stream().mapToInt(p -> p[1]).min().getAsInt();
275         int max = positions.stream().mapToInt(p -> p[1]).max().getAsInt();
276         if (max - min + 1 != positions.size()) {
277             throw new IllegalArgumentException("Vehicle '" + c + "' is not contiguous (has a gap).");
278         }
279     } else { // sameCol
280         int min = positions.stream().mapToInt(p -> p[0]).min().getAsInt();
281         int max = positions.stream().mapToInt(p -> p[0]).max().getAsInt();
282         if (max - min + 1 != positions.size()) {
283             throw new IllegalArgumentException("Vehicle '" + c + "' is not contiguous (has a gap).");
284         }
285     }
286
287     if (c == 'P') {
288         if (foundP) {
289             throw new IllegalArgumentException(s:"There must be exactly one primary piece. None found.");
290         }
291         foundP = true;
292     }
293 }
294
295
296 if (!foundP) {
297     throw new IllegalArgumentException(s:"There must be exactly one primary piece. None was found.");
298 }
299
300 }
301

```

## 4.9.GUIApp.java

```
GUIApp.java X
src > gui > GUIApp.java > Language Support for Java(TM) by Red Hat > {} gui
1  package gui;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileWriter;
6  import java.io.PrintWriter;
7  import java.util.HashMap;
8  import java.util.List;
9  import java.util.Map;
10 import javafx.animation.KeyFrame;
11 import javafx.animation.Timeline;
12 import javafx.application.Application;
13 import javafx.geometry.Insets;
14 import javafx.geometry.Pos;
15 import javafx.scene.Scene;
16 import javafx.scene.control.*;
17 import javafx.scene.layout.*;
18 import javafx.scene.paint.Color;
19 import javafx.scene.text.Font;
20 import javafx.scene.text.Text;
21 import javafx.stage.FileChooser;
22 import javafx.stage.Stage;
23 import javafx.util.Duration;
24 import model.Board;
25 import util.InputParser;
26 import algo.*;
27
28 public class GUIApp extends Application {
29     private List<Board> solution;
30     private int currentStep = 0;
31     private Timeline animationTimeline;
32     private Solver solver;
33     private final Map<Character, Color> pieceColors = new HashMap<>();
34
35     private final GridPane boardGrid = new GridPane();
36     private final Label stepLabel = new Label(text:"Step 0 / 0");
37     private final TextArea outputArea = new TextArea();
38     private VBox rightPanel = new VBox(spacing:10);
39     private final Label nodeVisitedLabel;
40     private final Label execTimeLabel = new Label();
41     private final Label movementBlock = new Label();
42     private final Label statusLabel = new Label();
43
44     public GUIApp() {
45         this.nodeVisitedLabel = new Label();
46     }
47
48     @Override
49     public void start(Stage primaryStage) {
50         primaryStage.setTitle(value:"Rush Hour Puzzle Solver");
51
52         initializeColorPalette();
```

```

53
54 // title
55 Font titleFont = Font.loadFont(getClass().getResourceAsStream(name:"/fonts/StayPlayful.ttf"), size:32);
56 Label title = new Label(text:"Rush Hour Puzzle Solver");
57 title.setFont(titleFont);
58 HBox titleBox = new HBox(title);
59 titleBox.setAlignment(Pos.CENTER);
60
61 // input file
62 Label fileLabel = new Label(text:"Select input configuration file:");
63 Button fileButton = new Button(text:"Browse...");
64 Label selectedFileLabel = new Label(text:"No file selected.");
65 HBox inputFileBox = new HBox(spacing:10, fileButton, selectedFileLabel);
66 FileChooser fileChooser = new FileChooser();
67 fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter(description:"Text Files", ...extensions:"*.txt"));
68
69 // algorithm selection
70 Label algoLabel = new Label(text:"Choose the algorithm:");
71 ComboBox<String> algoComboBox = new ComboBox<>();
72 algoComboBox.getItems().addAll(...elements:"A*", "UCS", "GBFS", "IDS");
73 algoComboBox.setValue(value:"A*");
74
75 // heuristic selection
76 Label heuristicLabel = new Label(text:"Heuristic:");
77 ComboBox<String> heuristicComboBox = new ComboBox<>();
78 heuristicComboBox.getItems().addAll(...elements:"Blocking Pieces Count", "Blocking Pieces With Movability", "Distance-to-Exit");
79 heuristicComboBox.setValue(value:"Blocking Pieces Count");
80
81 // ids depth input
82 Label depthLabel = new Label(text:"IDS Max Depth:");
83 TextField depthInput = new TextField(text:"20");
84 depthLabel.setVisible(value:false);
85 depthInput.setVisible(value:false);
86
87 // output format selection
88 Label outputTypeLabel = new Label(text:"Output:");
89 ComboBox<String> outputTypeComboBox = new ComboBox<>();
90 outputTypeComboBox.getItems().addAll(...elements:"Pagination", "Animation");
91 outputTypeComboBox.setValue(value:"Pagination");
92
93 // animation delay input
94 Label delayLabel = new Label(text:"Animation delay (ms):");
95 TextField delayInput = new TextField(text:"500");
96 delayLabel.setVisible(value:false);
97 delayInput.setVisible(value:false);
98
99 Button solveButton = new Button(text:"Solve");
100 Button exportButton = new Button(text:"Save Solution");
101
102 // controls
103 Button prevButton = new Button(text:"Previous");
104 Button nextButton = new Button(text:"Next");
105 Button finalButton = new Button(text:"Final");

```

```

106     prevButton.setVisible(value:false);
107     nextButton.setVisible(value:false);
108     finalButton.setVisible(value:false);
109     stepLabel.setVisible(value:false);
110     exportButton.setVisible(value:false);
111
112
113     // input panel
114     VBox inputPanel = new VBox(spacing:10);
115     inputPanel.getChildren().addAll(
116         fileLabel, inputFileBox, algoLabel, algoComboBox,
117         depthLabel, depthInput,
118         heuristicLabel, heuristicComboBox,
119         outputTypeLabel, outputTypeComboBox,
120         delayLabel, delayInput,
121         solveButton, exportButton
122     );
123     inputPanel.setPadding(new Insets(topRightBottomLeft:10));
124
125     outputArea.setPrefWidth(value:300);
126     outputArea.setEditable(value:false);
127     outputArea.setStyle(value:"-fx-font-family: monospace;");
128
129     HBox controls = new HBox(spacing:10, prevButton, stepLabel, nextButton, finalButton);
130
131     ToggleButton toggleLogButton = new ToggleButton(text:"Show Logs");
132     toggleLogButton.setOnAction(ev -> {
133         if (toggleLogButton.isSelected()) {
134             toggleLogButton.setText(value:"Hide Logs");
135             if (!rightPanel.getChildren().contains(outputArea)) {
136                 rightPanel.getChildren().add(outputArea);
137             }
138         } else {
139             toggleLogButton.setText(value:"Show Logs");
140             rightPanel.getChildren().remove(outputArea);
141         }
142     });
143
144     // right panel
145     rightPanel.getChildren().addAll(
146         new Label(text:"Board:"),
147         boardGrid,
148         controls,
149         movementBlock,
150         nodeVisitedLabel,
151         execTimeLabel,
152         statusLabel,
153         toggleLogButton
154     );
155     rightPanel.setPadding(new Insets(topRightBottomLeft:10));
156

```



```

157 ScrollPane rightScrollPane = new ScrollPane(rightPanel);
158 HBox.setHgrow(rightScrollPane, Priority.ALWAYS);
159 rightScrollPane.setMaxWidth(Double.MAX_VALUE);
160 rightScrollPane.setFitToWidth(value:true);
161 rightScrollPane.setPrefViewportHeight(value:600);
162
163 // main panel
164 HBox mainPanels = new HBox(spacing:20, inputPanel, rightScrollPane);
165
166 VBox root = new VBox(spacing:10, titleBox, mainPanels);
167 root.setPadding(new Insets(topRightBottomLeft:10));
168
169 Scene scene = new Scene(root, width:900, height:600);
170 primaryStage.setScene(scene);
171 primaryStage.show();
172
173 final Board[] board = new Board[1];
174
175 fileButton.setOnAction(e -> {
176     File file = fileChooser.showOpenDialog(primaryStage);
177     if (file != null) {
178         selectedFileLabel.setText(file.getName());
179         stepLabel.setVisible(value:false);
180         prevButton.setVisible(value:false);
181         nextButton.setVisible(value:false);
182         finalButton.setVisible(value:false);
183         movementBlock.setVisible(value:false);
184         exportButton.setVisible(value:false);
185         nodeVisitedLabel.setVisible(value:false);
186         execTimeLabel.setVisible(value:false);
187         try {
188             board[0] = InputParser.parse(file);
189             outputArea.setText(value:"File loaded successfully.\n");
190             statusLabel.setText(value:"");
191             drawBoard(board[0]);
192             solveButton.setDisable(value:false);
193         } catch (FileNotFoundException ex) {
194             boardGrid.getChildren().clear();
195             statusLabel.setVisible(value:true);
196             outputArea.setText("Failed to read the file: " + ex.getMessage());
197             statusLabel.setText("Failed to read the file: " + ex.getMessage());
198             solveButton.setDisable(value:true);
199         } catch (IllegalArgumentException ex) {
200             boardGrid.getChildren().clear();
201             statusLabel.setVisible(value:true);
202             outputArea.setText("Error: " + ex.getMessage());
203             statusLabel.setText("Error: " + ex.getMessage());
204             solveButton.setDisable(value:true);
205         }
206     }
207 });

```

```

208
209     algoComboBox.setOnAction(e -> {
210         String selectedAlgo = algoComboBox.getValue();
211         heuristicComboBox.setDisable(!selectedAlgo.equals(anObject:"A*") && !selectedAlgo.equals(anObject:"GBFS"));
212         depthLabel.setVisible(selectedAlgo.equals(anObject:"IDS"));
213         depthInput.setVisible(selectedAlgo.equals(anObject:"IDS"));
214     });
215
216     outputTypeComboBox.setOnAction(e -> {
217         boolean isAnimation = outputTypeComboBox.getValue().equals(anObject:"Animation");
218         delayLabel.setVisible(isAnimation);
219         delayInput.setVisible(isAnimation);
220     });
221
222     solveButton.setOnAction(e -> {
223         if (board[0] == null) {
224             outputArea.setText(value:"Please load a puzzle file.");
225             return;
226         }
227
228         switch (algoComboBox.getValue()) {
229             case "A*" -> solver = new AStarSolver(heuristicComboBox.getValue());
230             case "UCS" -> solver = new UCSolver();
231             case "GBFS" -> solver = new GBFSSolver(heuristicComboBox.getValue());
232             case "IDS" -> {
233                 try {
234                     int maxDepth = Integer.parseInt(depthInput.getText());
235                     solver = new IDSSolver(maxDepth);
236                 } catch (NumberFormatException ex) {
237                     outputArea.setText(value:"Invalid IDS max depth.");
238                     statusLabel.setText(value:"Invalid IDS max depth.");
239                     return;
240                 }
241             }
242             default -> {
243                 outputArea.setText(value:"Unknown algorithm.");
244                 return;
245             }
246         }
247

```

```

248     solution = solver.solve(board[0]);
249     if (solution.isEmpty()) {
250         Alert alert = new Alert(Alert.AlertType.INFORMATION);
251         alert.setTitle(title:"No Solution");
252         alert.setHeaderText(headerText:null);
253         alert.setContentText(contentText:"No solution was found for this puzzle.");
254         alert.showAndWait();
255         nodeVisitedLabel.setText(value:"");
256         execTimeLabel.setText(value:"");
257         stepLabel.setVisible(value:false);
258         movementBlock.setVisible(value:false);
259         prevButton.setVisible(value:false);
260         nextButton.setVisible(value:false);
261         finalButton.setVisible(value:false);
262         exportButton.setVisible(value:false);
263         statusLabel.setVisible(value:false);
264         return;
265     }
266
267     currentStep = 0;
268     stepLabel.setVisible(value:true);
269     prevButton.setVisible(value:true);
270     nextButton.setVisible(value:true);
271     finalButton.setVisible(value:true);
272     exportButton.setVisible(value:true);
273     movementBlock.setVisible(value:true);
274     nodeVisitedLabel.setVisible(value:true);
275     execTimeLabel.setVisible(value:true);
276     statusLabel.setVisible(value:true);
277
278     String mode = outputTypeComboBox.getValue();
279     if (mode.equals(anObject:"Pagination")) {
280         displayStep(currentStep);
281     } else {
282         int delay = Integer.parseInt(delayInput.getText());
283         animationTimeline = new Timeline(new KeyFrame(Duration.millis(delay), ev -> {
284             if (currentStep < solution.size()) {
285                 displayStep(currentStep);
286
287                 Board current = solution.get(currentStep);
288                 if (current.move != null && current.move.equals(anObject:"Primary piece exits through K")) {
289                     currentStep = solution.size() - 1;
290                     displayStep(currentStep);
291                     animationTimeline.stop();
292                 } else {
293                     currentStep++;
294                 }
295             } else {
296                 animationTimeline.stop();
297             }
298         }));
299         animationTimeline.setCycleCount(Timeline.INDEFINITE);
300         animationTimeline.play();

```

```

301     }
302 });
303
304 prevButton.setOnAction(e -> {
305     if (currentStep > 0) displayStep(--currentStep);
306 });
307
308 nextButton.setOnAction(e -> {
309     if (currentStep < solution.size() - 1) {
310         Board current = solution.get(currentStep);
311         if (current.move != null && current.move.equals(anObject:"Primary piece exits through K")) {
312             currentStep = solution.size() - 1;
313         } else {
314             currentStep++;
315         }
316         displayStep(currentStep);
317     }
318 });
319
320 finalButton.setOnAction(e -> {
321     currentStep = solution.size() - 1;
322     displayStep(currentStep);
323 });
324
325 exportButton.setOnAction(e -> {
326     FileChooser chooser = new FileChooser();
327     chooser.setTitle(value:"Export Solution");
328     chooser.getExtensionFilters().add(new FileChooser.ExtensionFilter(description:"Text Files", ...extensions:"*.txt"));
329     chooser.setInitialFileName(value:"solution.txt");
330
331     File file = chooser.showSaveDialog(primaryStage);
332     if (file != null) {
333         if (!file.getName().toLowerCase().endsWith(suffix:".txt")) {
334             file = new File(file.getParentFile(), file.getName() + ".txt");
335         }
336
337         try (PrintWriter writer = new PrintWriter(new FileWriter(file))) {
338             for (int i = 0; i < solution.size(); i++) {
339                 Board current = solution.get(i);
340                 if (i == 0) {
341                     writer.println(x:"Initial state");
342                     writer.println("Step " + i + ":");
343                 } else {
344                     writer.println("Step " + i + ":");
345                     String move = current.move;
346                     if (move != null) {
347                         String[] parts = move.split(regex:" ");
348                         if (parts.length == 3) {
349                             char piece = parts[1].charAt(index:0);
350                             String direction = parts[2];
351                             writer.println("Block " + piece + " moved " + direction);
352                         }
353                     }
354                 }
355             }
356         }
357     }
358 }

```

```

354         }
355
356         writer.println(current.toString());
357         writer.println();
358     }
359
360     writer.println("Nodes Visited: " + solver.getVisitedNodeCount());
361     writer.println("Execution Time: " + solver.getExecutionTime() + " ms");
362
363     outputArea.appendText("\nExported to " + file.getName());
364     statusLabel.setText("Exported to " + file.getName());
365 } catch (Exception ex) {
366     outputArea.appendText("\nFailed to export: " + ex.getMessage());
367     statusLabel.setText("Failed to export: " + ex.getMessage());
368 }
369 }
370 });
371 }
372
373 private void initializeColorPalette() {
374     List<Color> palette = List.of(
375         Color.web(colorString: "#e6194b"), // red (for P)
376         Color.web(colorString: "#3cb44b"), // green (for K)
377         Color.web(colorString: "#ffe119"), // yellow
378         Color.web(colorString: "#4363d8"), // blue
379         Color.web(colorString: "#f58231"), // orange
380         Color.web(colorString: "#911eb4"), // purple
381         Color.web(colorString: "#46f0f0"), // cyan
382         Color.web(colorString: "#f032e6"), // magenta
383         Color.web(colorString: "#bcb6c1"), // lime
384         Color.web(colorString: "#fabebe"), // pink
385         Color.web(colorString: "#008080"), // teal
386         Color.web(colorString: "#e6beff"), // lavender
387         Color.web(colorString: "#9a6324"), // brown
388         Color.web(colorString: "#fffac8"), // beige
389         Color.web(colorString: "#800000"), // maroon
390         Color.web(colorString: "#aaffc3"), // mint
391         Color.web(colorString: "#808000"), // olive
392         Color.web(colorString: "#ffd8b1"), // coral
393         Color.web(colorString: "#000075"), // navy
394         Color.web(colorString: "#808080") // gray
395     );
396
397     int paletteIndex = 0;
398     for (char c = 'A'; c <= 'Z'; c++) {
399         if (c != 'P' && c != 'K') {
400             pieceColors.put(c, palette.get(paletteIndex % palette.size()));
401             paletteIndex++;
402         }
403     }
404     pieceColors.put(key: 'P', Color.RED);
405     pieceColors.put(key: 'K', Color.GREEN);
406 }

```

```

408 private void drawBoard(Board board) {
409     boardGrid.getChildren().clear();
410     boardGrid.setGridLinesVisible(value:true);
411     char[][] grid = board.getGrid();
412     int rows = board.getRows(), cols = board.getCols();
413
414     boolean shiftRight = board.getExitCol() == -1;
415     boolean shiftDown = board.getExitRow() == -1;
416
417     boardGrid.getChildren().clear();
418     boardGrid.getColumnConstraints().clear();
419     boardGrid.getRowConstraints().clear();
420
421     for (int j = 0; j < cols + (shiftRight ? 1 : 0); j++) {
422         ColumnConstraints col = new ColumnConstraints(width:50);
423         boardGrid.getColumnConstraints().add(col);
424     }
425
426     for (int i = 0; i < rows + (shiftDown ? 1 : 0); i++) {
427         RowConstraints row = new RowConstraints(height:50);
428         boardGrid.getRowConstraints().add(row);
429     }
430
431     for (int i = 0; i < rows; i++) {
432         for (int j = 0; j < cols; j++) {
433             char ch = grid[i][j];
434             StackPane cell = new StackPane();
435             cell.setPrefSize(prefWidth:50, prefHeight:50);
436             cell.setStyle(value:"-fx-border-color: black;");
437             if (ch != '.') {
438                 if (ch != 'P' || !board.isGoal()) {
439                     Color color = pieceColors.getOrDefault(ch, Color.GREY);
440                     cell.setBackground(new Background(new BackgroundFill(color, CornerRadii.EMPTY, Insets.EMPTY)));
441                     Text label = new Text(String.valueOf(ch));
442                     label.setFont(Font.font(size:20));
443                     label.setFill(Color.BLACK);
444                     cell.getChildren().add(label);
445                 }
446             }
447             boardGrid.add(cell, j + (shiftRight ? 1 : 0), i + (shiftDown ? 1 : 0));
448         }
449     }
450
451     int exitRow = board.getExitRow();
452     int exitCol = board.getExitCol();
453
454     if (exitRow >= -1 && exitRow <= rows && exitCol >= -1 && exitCol <= cols) {
455         StackPane exitCell = new StackPane();
456         exitCell.setPrefSize(prefWidth:50, prefHeight:50);
457         exitCell.setBackground(new Background(new BackgroundFill(Color.LIGHTGRAY, CornerRadii.EMPTY, Insets.EMPTY)));
458
459         BorderWidths borderWidths;

```

```

460
461     if (exitRow == -1) {
462         borderWidths = new BorderWidths(top:1, right:1, bottom:0, left:1);
463     } else if (exitRow == rows) {
464         borderWidths = new BorderWidths(top:0, right:1, bottom:1, left:1);
465     } else if (exitCol == -1) {
466         borderWidths = new BorderWidths(top:1, right:0, bottom:1, left:1);
467     } else if (exitCol == cols) {
468         borderWidths = new BorderWidths(top:1, right:1, bottom:1, left:0);
469     } else {
470         borderWidths = new BorderWidths(width:0);
471     }
472
473     exitCell.setBorder(new Border(new BorderStroke(
474         Color.BLACK,
475         BorderStrokeStyle.SOLID,
476         CornerRadii.EMPTY,
477         borderWidths
478     )));
479
480     Text label = new Text(text:"K");
481     label.setFont(Font.font(size:20));
482     label.setFill(Color.BLACK);
483     exitCell.getChildren().add(label);
484
485     int gridRow = exitRow + (shiftDown ? 1 : 0);
486     int gridCol = exitCol + (shiftRight ? 1 : 0);
487
488     if (exitCol == cols) {
489         if (boardGrid.getColumnConstraints().size() <= cols + (shiftRight ? 1 : 0)) {
490             boardGrid.getColumnConstraints().add(new ColumnConstraints(width:50));
491         }
492     } else if (exitCol == -1) {
493         boardGrid.getColumnConstraints().add(index:0, new ColumnConstraints(width:50));
494     }
495
496     if (exitRow == rows) {
497         if (boardGrid.getRowConstraints().size() <= rows + (shiftDown ? 1 : 0)) {
498             boardGrid.getRowConstraints().add(new RowConstraints(height:50));
499         }
500     } else if (exitRow == -1) {
501         boardGrid.getRowConstraints().add(index:0, new RowConstraints(height:50));
502     }
503
504     boardGrid.add(exitCell, gridCol, gridRow);
505 }
506 }
507

```

```

508 private void displayStep(int step) {
509     Board current = solution.get(step);
510     drawBoard(current);
511
512     StringBuilder sb = new StringBuilder();
513     sb.append(str:"Step ").append(step).append(str:":\n");
514
515     // default label text
516     String movementText = "Initial state";
517
518     // tampilkan info gerakan jika bukan board awal
519     if (step > 0) {
520         String move = current.move;
521         if (move != null) {
522             if (move.equals(anObject:"Primary piece P exits through K")) {
523                 sb.append(str:"Primary piece P exited the board through the exit!\n");
524                 movementText = "Primary piece P exited the board!";
525             } else {
526                 String[] parts = move.split(regex:" ");
527                 if (parts.length == 3) {
528                     char piece = parts[1].charAt(index:0);
529                     String direction = parts[2];
530
531                     sb.append(str:"Block ").append(piece).append(str:" moved ").append(direction).append(str:"\n");
532                     movementText = "Block " + piece + " moved " + direction;
533                 }
534             }
535         }
536     } else {
537         sb.append(str:"Initial state\n");
538     }
539
540     sb.append(current.toString());
541
542     outputArea.setText(sb.toString());
543     movementBlock.setText(movementText);
544     stepLabel.setText("Step " + step + " / " + (solution.size() - 1));
545     nodeVisitedLabel.setText("Nodes Visited: " + solver.getVisitedNodeCount());
546     execTimeLabel.setText("Execution Time: " + solver.getExecutionTime() + " ms");
547 }
548
549 Run main | Debug main | Run | Debug
550 public static void main(String[] args) {
551     launch(args);
552 }
553
554 public VBox getRightPanel() {
555     return rightPanel;
556 }
557
558 public void setRightPanel(VBox rightPanel) {
559     this.rightPanel = rightPanel;
560 }

```



## BAB V

### PENGUJIAN MASUKAN DAN LUARAN PROGRAM

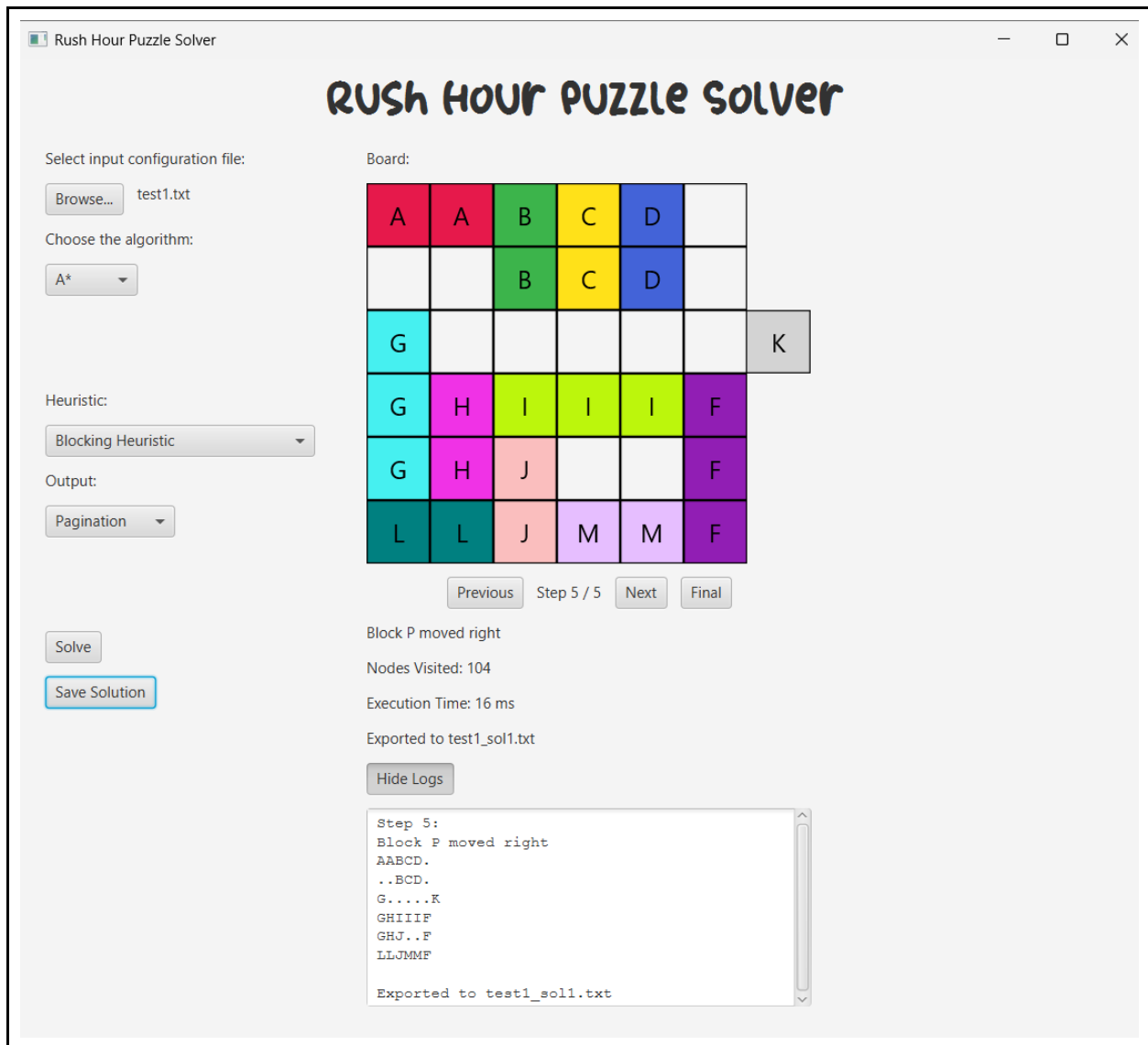
Berikut merupakan tangkapan layar yang memperlihatkan *input* dan *output test case*.

#### 5.1.Test Case 1

Isi test1.txt	
<pre>test &gt; test1.txt 1 6 6 2 11 3 AAB..F 4 ..BCDF 5 GPPCDFK 6 GH.III 7 GHJ... 8 LLJMM.</pre>	

Tampilan pertama saat program baru dijalankan	
	

Output penyelesaian program test1.txt pada GUI
--



*Output penyelesaian program test1.txt yang telah disimpan pada file test1\_sol1.txt*

Initial state

Step 0:

AAB..F

..BCDF

GPPCDFK

GH.III

GHJ...

LLJMM.

Step 1:

Block I moved left

AAB..F  
..BCDF  
GPPCDFK  
GHII.  
GHJ...  
LLJMM.

Step 2:  
Block C moved up  
AABC.F  
..BCDF  
GPP.DFK  
GHII.  
GHJ...  
LLJMM.

Step 3:  
Block F moved down  
AABC..  
..BCD.  
GPP.D.K  
GHIIF  
GHJ..F  
LLJM MF

Step 4:  
Block D moved up  
AABCD.  
..BCD.  
GPP...K  
GHIIF  
GHJ..F  
LLJM MF

Step 5:  
Block P moved right  
AABCD.  
..BCD.  
G.....K  
GHIIF  
GHJ..F  
LLJM MF

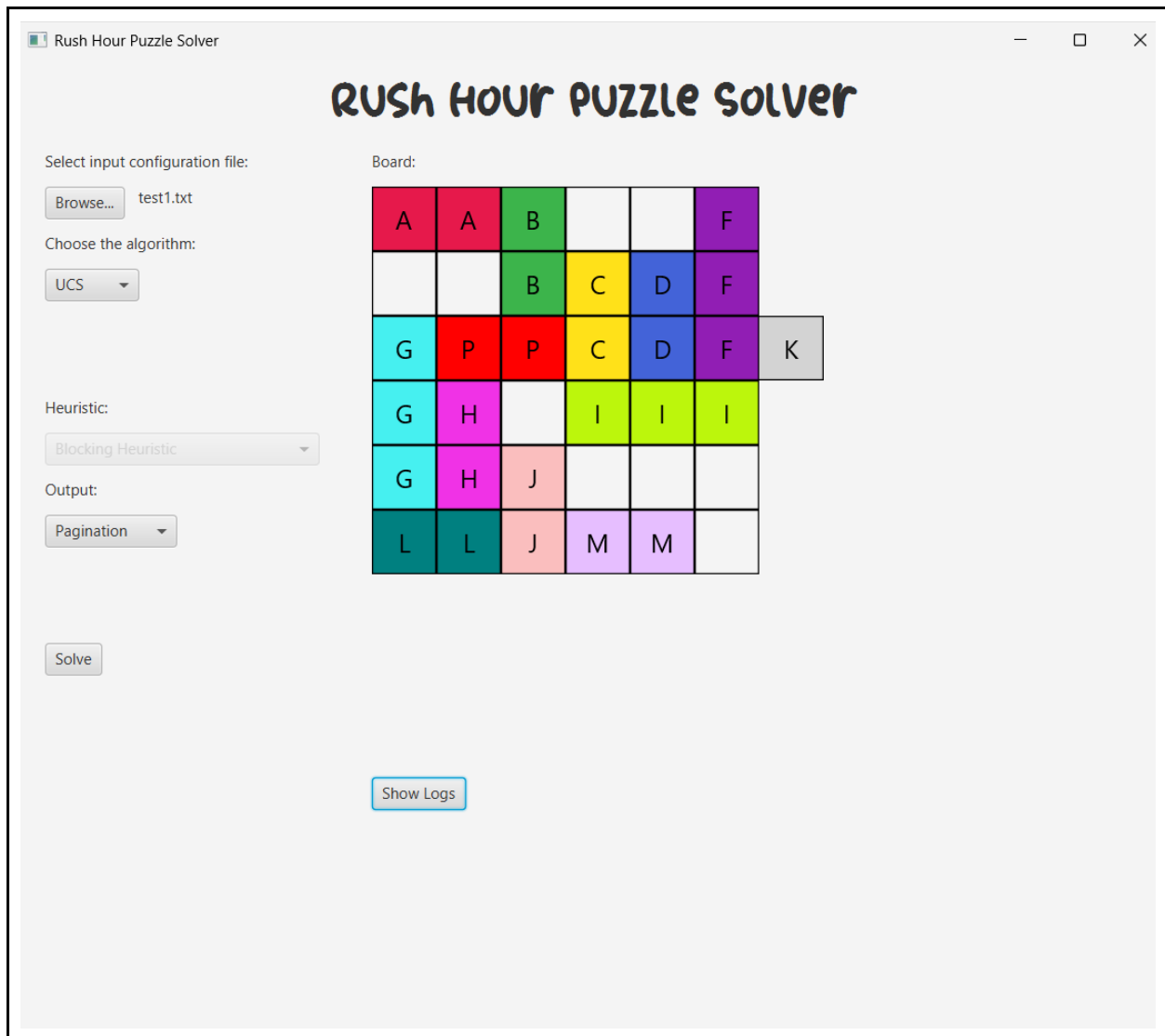
Nodes Visited: 104  
Execution Time: 16 ms

## 5.2. Test Case 2

Isi test1.txt

```
test > test1.txt
1 6 6
2 11
3 AAB..F
4 ..BCDF
5 GPPCDFK
6 GH.III
7 GHJ...
8 LLJMM.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test1.txt pada GUI*

Rush Hour Puzzle Solver

Select input configuration file:

Browse... test1.txt

Choose the algorithm:

UCS

Heuristic:

Blocking Heuristic

Output:

Pagination

Solve

Save Solution

Board:

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

Previous

Step 5 / 5

Next

Final

Block P moved right

Nodes Visited: 527

Execution Time: 34 ms

Exported to test1\_sol2.txt

Hide Logs

Step 5:

Block P moved right

AABCD.

..BCD.

G.....K

GHIIF

GHJ..F

LLJMMF

Exported to test1\_sol2.txt

*Output penyelesaian program test1.txt yang telah disimpan pada file test1\_sol2.txt*

Initial state

Step 0:

AAB..F

..BCDF

GPPCDFK

GH.III

GHJ...

LLJMM.

Step 1:

Block D moved up

AAB.DF  
..BCDF  
GPPC.FK  
GH.III  
GHJ...  
LLJMM.

Step 2:  
Block I moved left  
AAB.DF  
..BCDF  
GPPC.FK  
GH.III.  
GHJ...  
LLJMM.

Step 3:  
Block F moved down  
AAB.D.  
..BCD.  
GPPC..K  
GH.IIF  
GHJ..F  
LLJMMF

Step 4:  
Block C moved up  
AABCD.  
..BCD.  
GPP...K  
GH.IIF  
GHJ..F  
LLJMMF

Step 5:  
Block P moved right  
AABCD.  
..BCD.  
G.....K  
GH.IIF  
GHJ..F  
LLJMMF

Nodes Visited: 527  
Execution Time: 34 ms

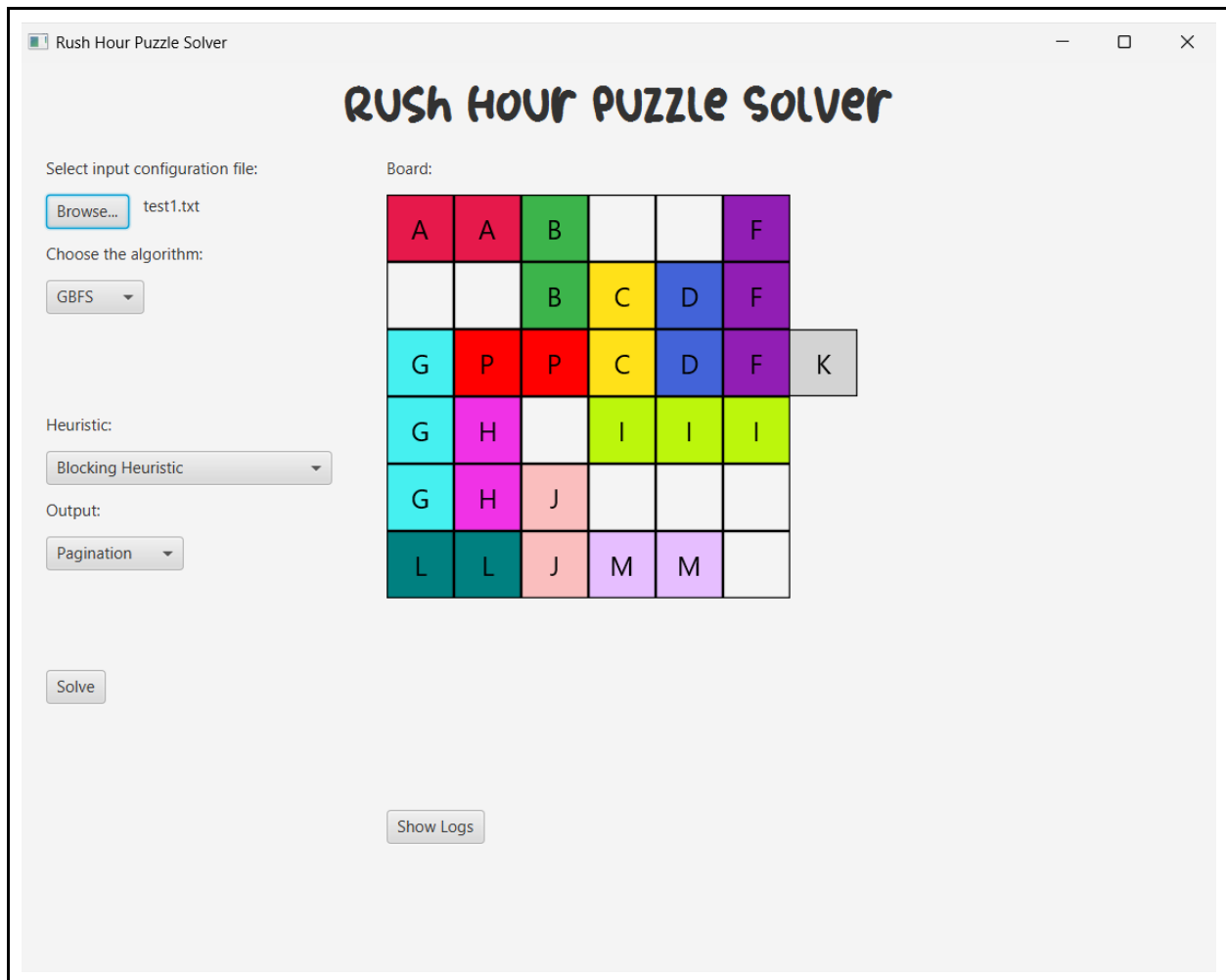
### 5.3.Test Case 3

Isi test1.txt

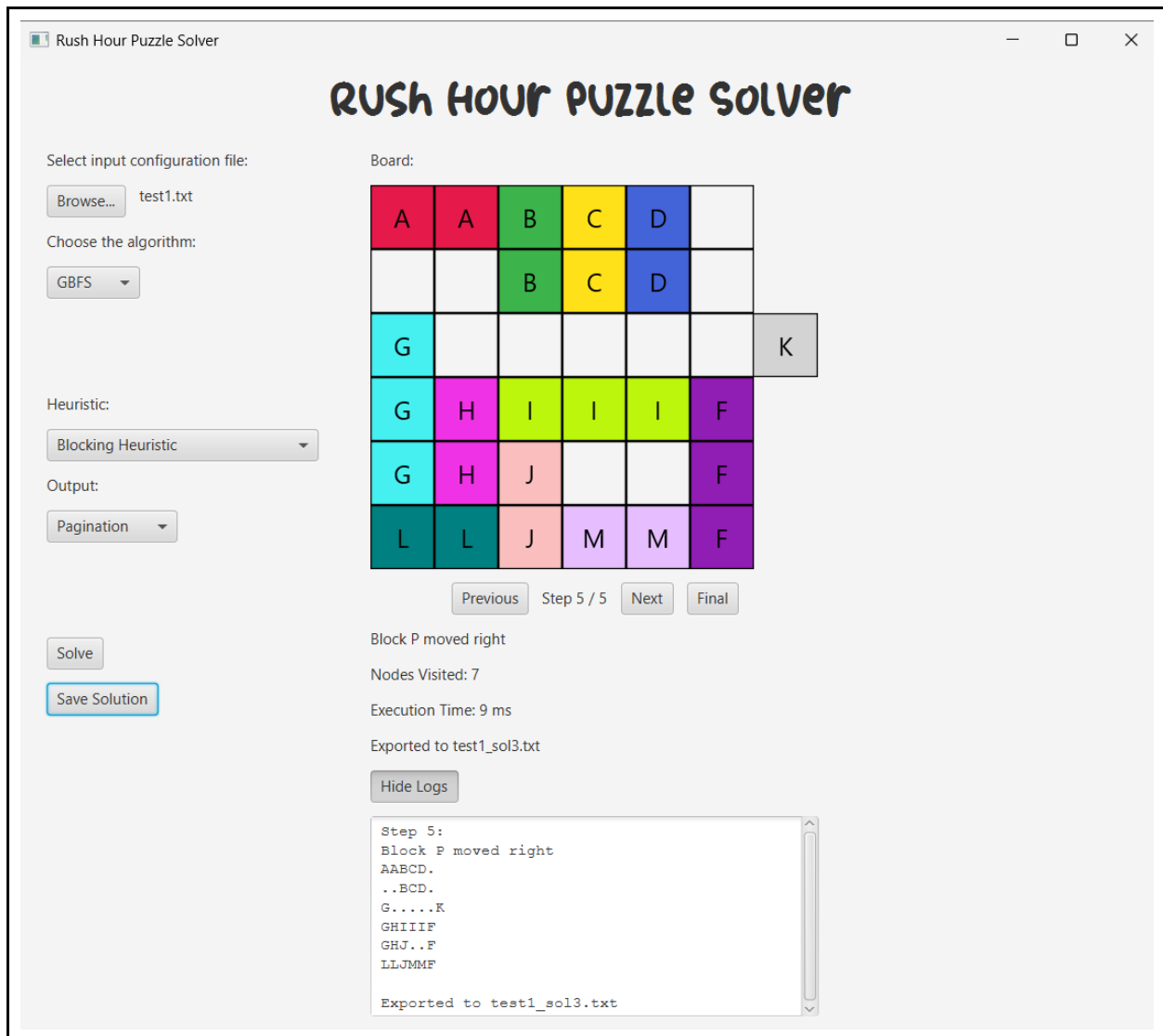
```
test > test1.txt
1 6 6
2 11
3 AAB..F
4 ..BCDF
5 GPPCDFK
6 GH.III
7 GHJ...
8 LLJMM.
```

Tampilan pertama saat program baru dijalankan





*Output penyelesaian program test1.txt pada GUI*



*Output penyelesaian program test1.txt yang telah disimpan pada file test1\_sol3.txt*

Initial state

Step 0:

AAB..F

..BCDF

GPPCDFK

GH.III

GHJ...

LLJMM.

Step 1:

Block C moved up

AABC.F  
..BCDF  
GPP.DFK  
GH.III  
GHJ...  
LLJMM.

Step 2:  
Block D moved up  
AABCDF  
..BCDF  
GPP..FK  
GH.III  
GHJ...  
LLJMM.

Step 3:  
Block I moved left  
AABCDF  
..BCDF  
GPP..FK  
GH.II.  
GHJ...  
LLJMM.

Step 4:  
Block F moved down  
AABCD.  
..BCD.  
GPP...K  
GHIIF  
GHJ..F  
LLJM MF

Step 5:  
Block P moved right  
AABCD.  
..BCD.  
G.....K  
GHIIF  
GHJ..F  
LLJM MF

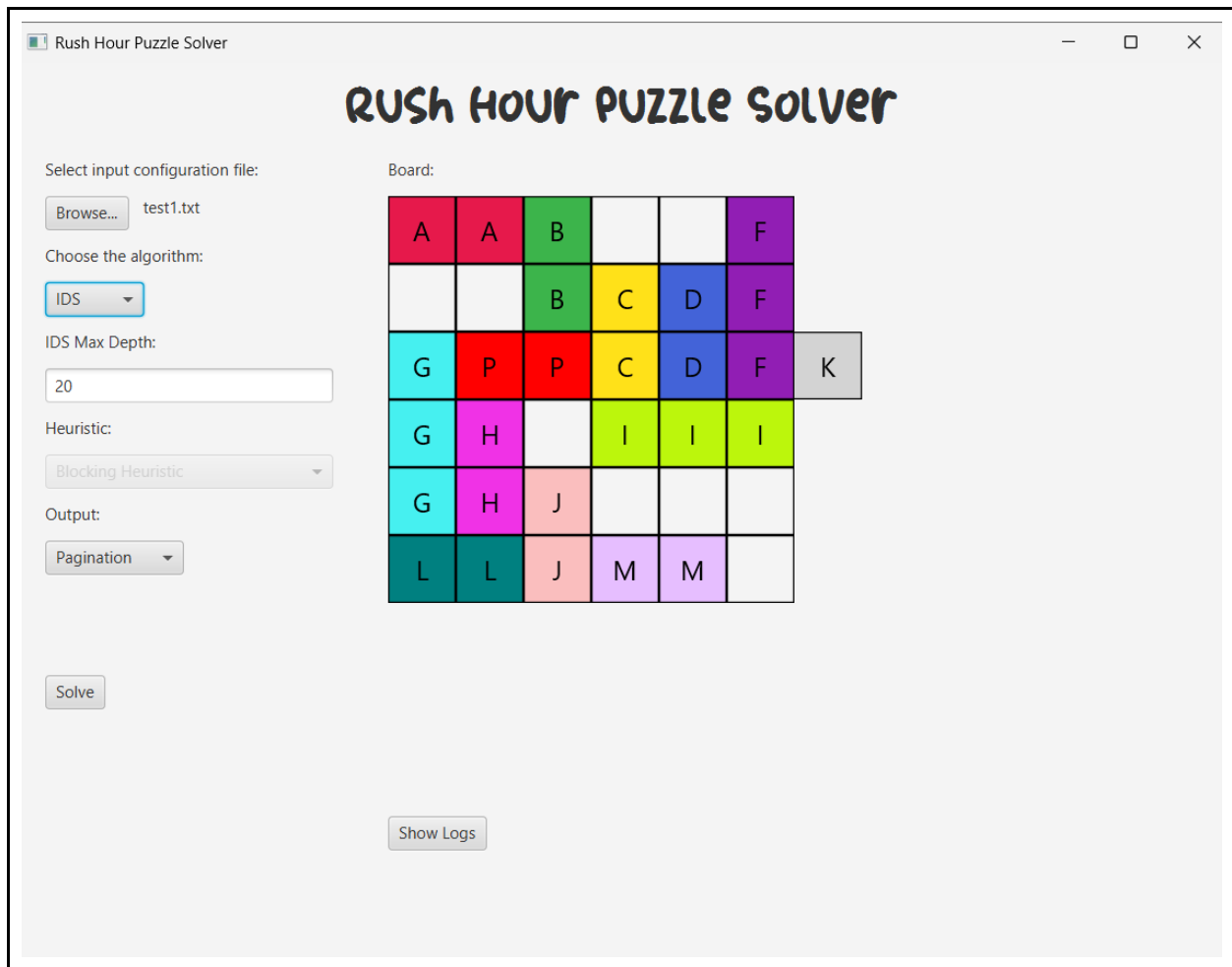
Nodes Visited: 7  
Execution Time: 9 ms

#### 5.4. Test Case 4

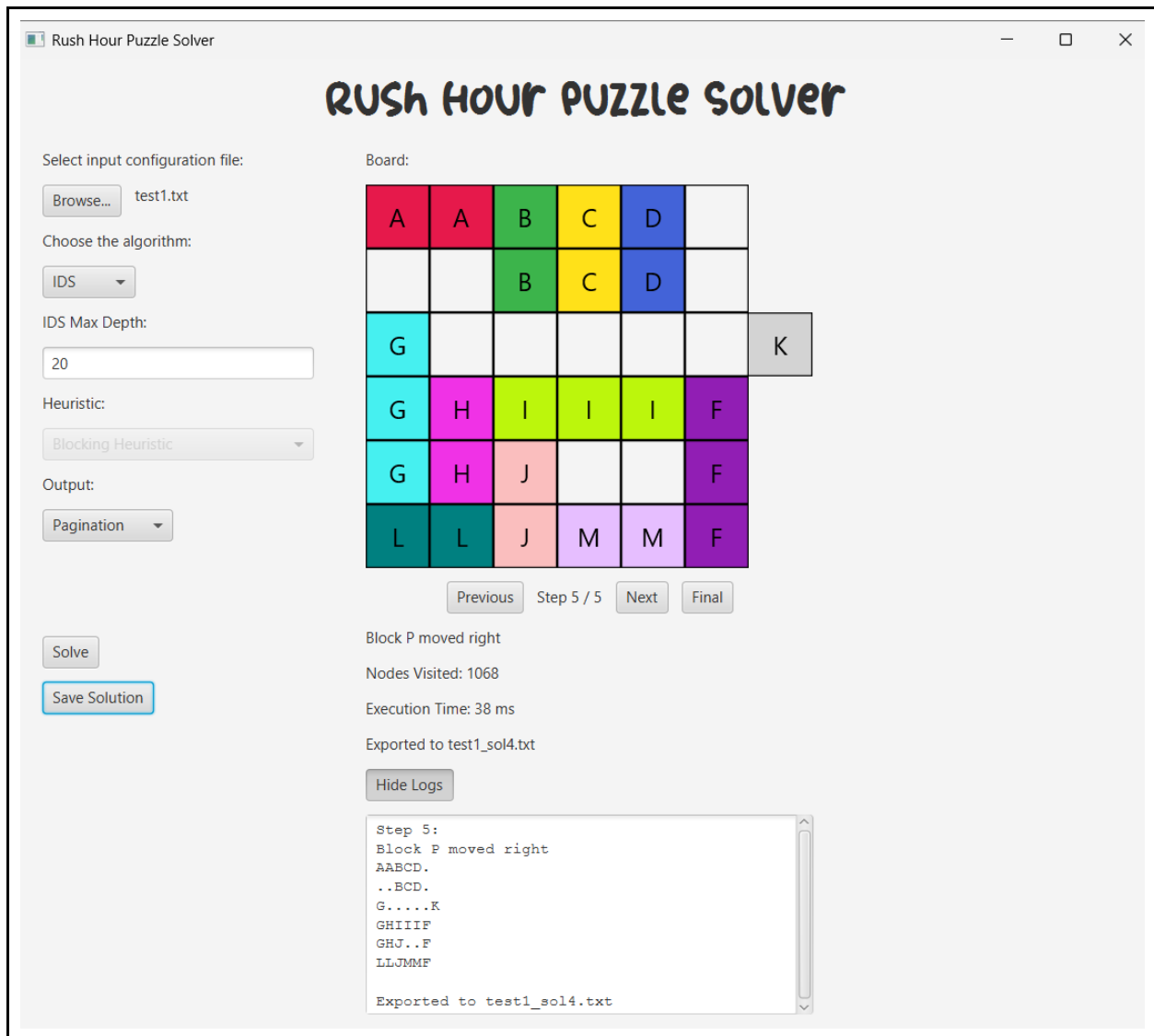
Isi test1.txt

```
test > test1.txt
1 6 6
2 11
3 AAB..F
4 ..BCDF
5 GPPCDFK
6 GH.III
7 GHJ...
8 LLJMM.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test1.txt pada GUI*



*Output penyelesaian program test1.txt yang telah disimpan pada file test1\_sol4.txt*

Initial state

Step 0:

AAB..F

..BCDF

GPPCDFK

GH.III

GHJ...

LLJMM.

Step 1:

Block C moved up

AABC.F  
..BCDF  
GPP.DFK  
GH.III  
GHJ...  
LLJMM.

Step 2:  
Block I moved left  
AABC.F  
..BCDF  
GPP.DFK  
GH.II.  
GHJ...  
LLJMM.

Step 3:  
Block F moved down  
AABC..  
..BCD.  
GPP.D.K  
GH.IIF  
GHJ..F  
LLJMMF

Step 4:  
Block D moved up  
AABCD.  
..BCD.  
GPP...K  
GH.IIF  
GHJ..F  
LLJMMF

Step 5:  
Block P moved right  
AABCD.  
..BCD.  
G.....K  
GH.IIF  
GHJ..F  
LLJMMF

Nodes Visited: 1068  
Execution Time: 38 ms

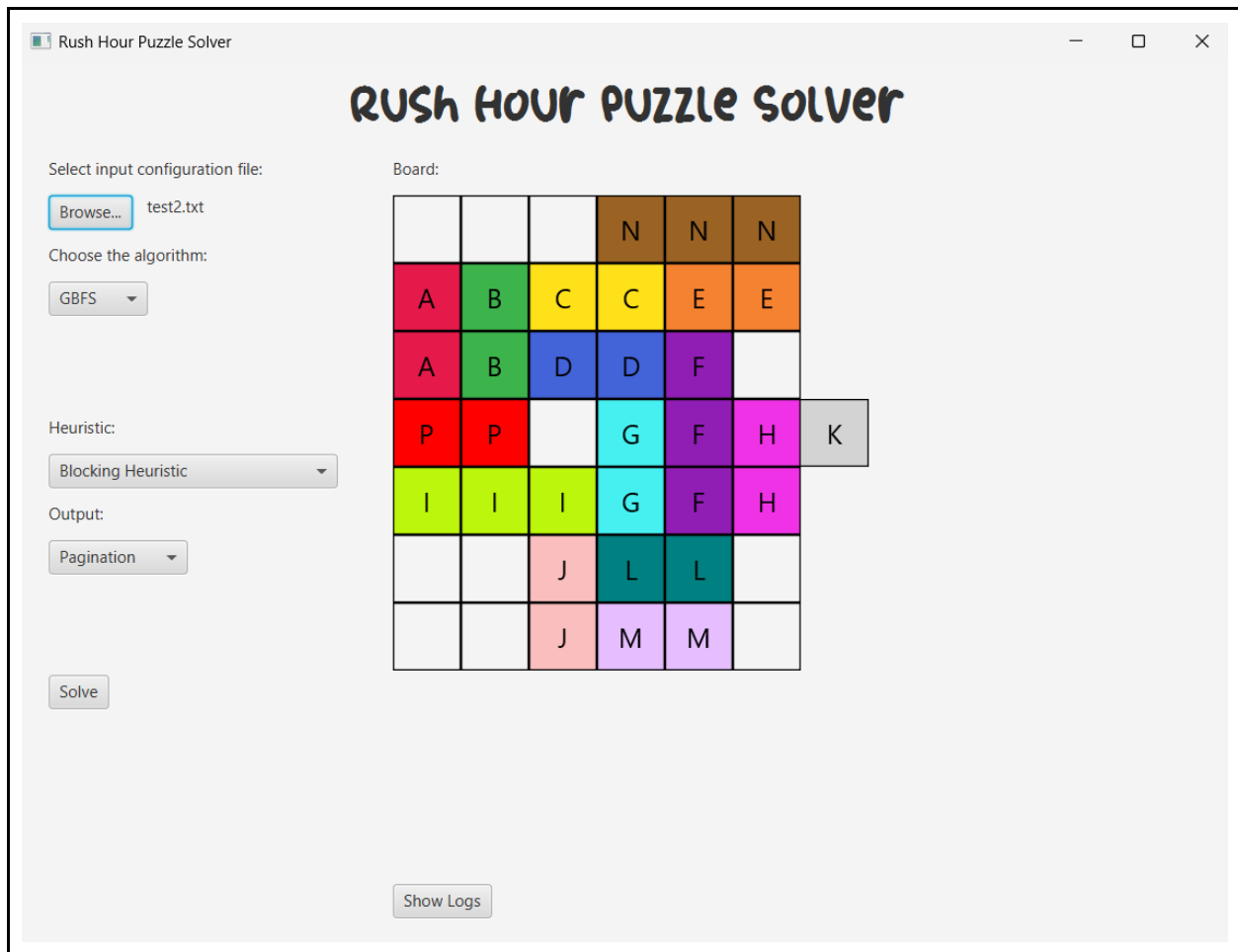
### 5.5.Test Case 5

Isi test2.txt

```
test > test2.txt
1 7 6
2 13
3 ...NNN
4 ABCCEE
5 ABDDF.
6 PP.GFHK
7 IIIGFH
8 ..JLL.
9 ..JMM.
```

Tampilan pertama saat program baru dijalankan





*Output penyelesaian program test2.txt pada GUI*

Rush Hour Puzzle Solver

Select input configuration file:

Browse...

test2.txt

Choose the algorithm:

GBFS

Heuristic:

Blocking Heuristic

Output:

Pagination

Solve

Save Solution

Board:

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

Previous

Step 71 / 71

Next

Final

Block P moved right

Nodes Visited: 2490

Execution Time: 118 ms

Exported to test2\_sol5.txt

Hide Logs

```

CCEEFH
ABDDFH
AB....K
.III..
..JGLL
..JGMM

Exported to test2_sol5.txt

```

*Output penyelesaian program test2.txt yang telah disimpan pada file test2\_sol5.txt*

Initial state

Step 0:

...NNN

ABCCEE

ABDDF.

PP.GFHK

IIIGFH

...JLL.

...JMM.

Step 1:

IF2211 Strategi Algoritma – Tugas Kecil 3

74

Block H moved down

...NNN

ABCCEE

ABDDF.

PP.GF.K

IIIGF.

..JLLH

..JMMH

.  
.  
.

Step 71:

Block P moved right

NNN.F.

CCEEFH

ABDDFH

AB....K

.III.

..JGLL

..JGMM

Nodes Visited: 2490

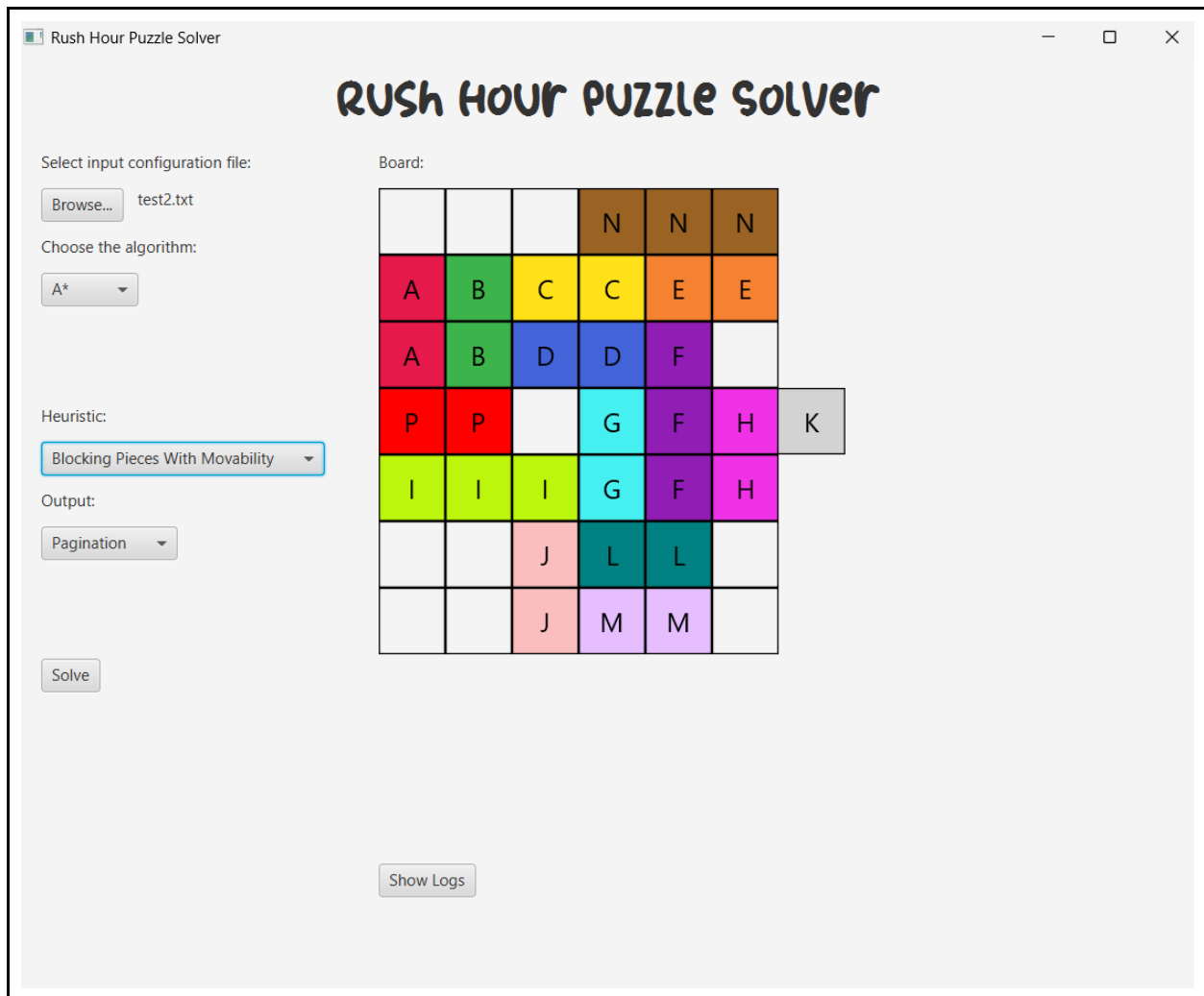
Execution Time: 118 ms

## 5.6.Test Case 6

Isi test2.txt

```
test > test2.txt
1 7 6
2 13
3 ...NNN
4 ABCCEE
5 ABDDF.
6 PP.GFHK
7 IIIGFH
8 ..JLL.
9 ..JMM.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test2.txt pada GUI*

Rush Hour Puzzle Solver

Select input configuration file:

Browse... test2.txt

Choose the algorithm:

A\*

Heuristic:

Blocking Pieces With Movability

Output:

Pagination

Solve

Save Solution

Board:

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

Previous

Step 11 / 11

Next

Final

Block P moved right

Nodes Visited: 2908

Execution Time: 233 ms

Exported to test2\_sol6.txt

Hide Logs

```

CCEEFH
ABDDF.
AB....K
IIIG..
..JGLL
..JMM.

Exported to test2_sol6.txt

```

*Output penyelesaian program test2.txt yang telah disimpan pada file test2\_sol6.txt*

Initial state

Step 0:

...NNN

ABCCEE

ABDDF.

PP.GFHK

IIIGFH

...JLL.

..JMM.

.

.

.

Step 11:  
Block P moved right  
NNN.FH  
CCEEFH  
ABDDF.  
AB....K  
IIIG..  
..JLL  
..JMM.

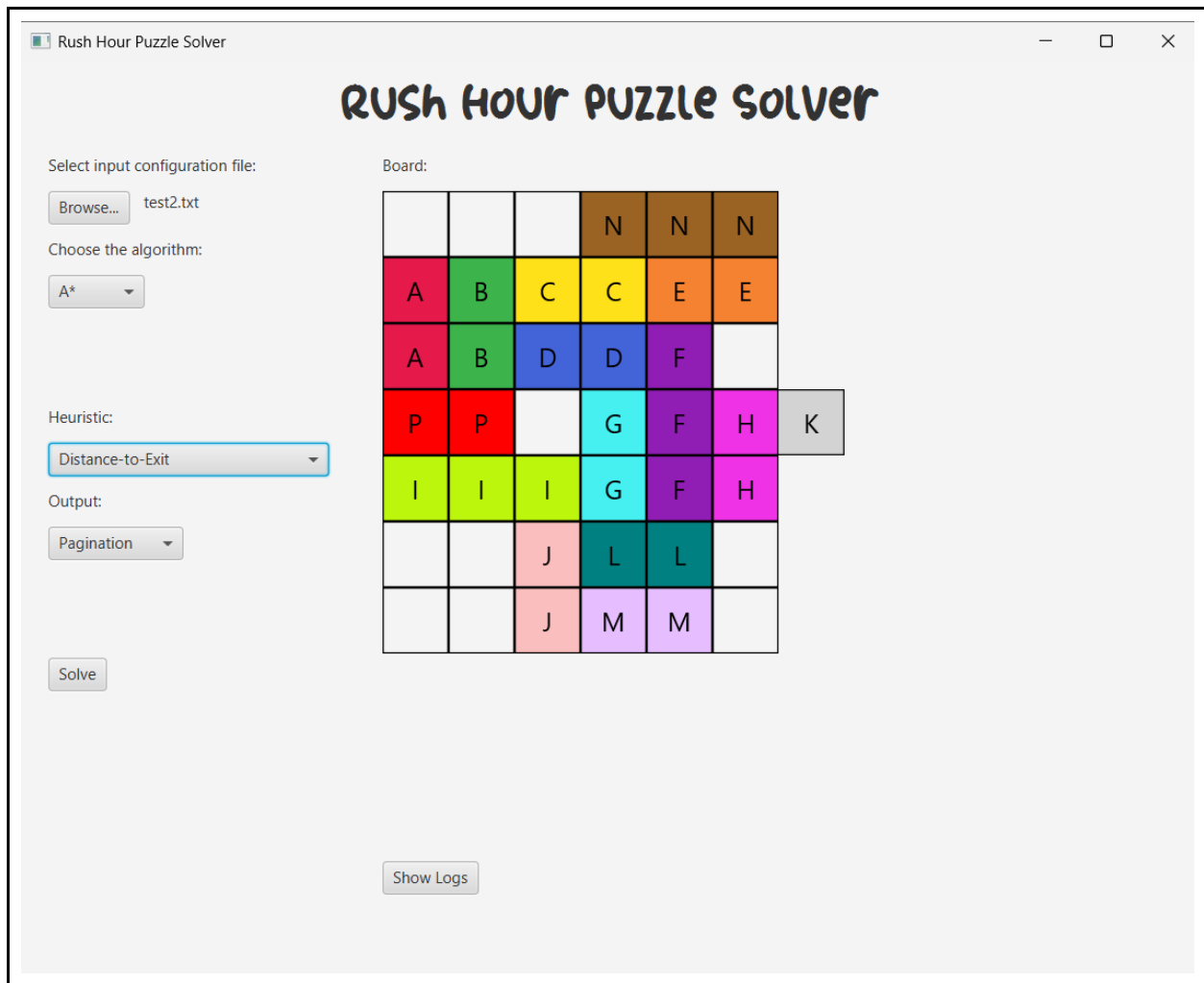
Nodes Visited: 2908  
Execution Time: 233 ms

### 5.7.Test Case 7

Isi test2.txt

```
test > test2.txt
1 7 6
2 13
3 ...NNN
4 ABCCEE
5 ABDDF.
6 PP.GFHK
7 IIIGFH
8 ..JLL.
9 ..JMM.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test2.txt pada GUI*

# RUSH HOUR PUZZLE SOLVER

Select input configuration file:

test2.txt

Choose the algorithm:

A\* ▼

Heuristic:

Distance-to-Exit ▼

Output:

Pagination ▼

Board:

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

Previous Step 11 / 11 Next Final

Block P moved right

Nodes Visited: 1433677

Execution Time: 23543 ms

Exported to test2\_sol7.txt

```

CCEEPH
ABDDF.
AB....K
IIIG..
..JGLL
..JMM.

Exported to test2_sol7.txt

```

*Output penyelesaian program test2.txt yang telah disimpan pada file test2\_sol7.txt*

Initial state

Step 0:

...NNN

ABCCEE

ABDDF.

PP.GFHK

IIIGFH

...JLL.

...JMM.

.

.



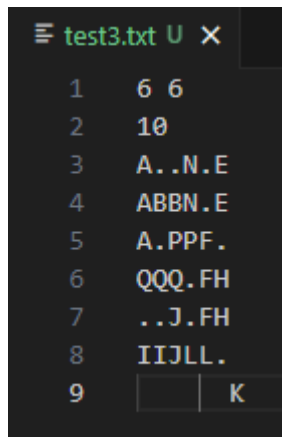
.

Step 11:  
Block P moved right  
NNN.FH  
CCEEFH  
ABDDF.  
AB....K  
IIIG..  
..JGLL  
..JMM.

Nodes Visited: 1433677  
Execution Time: 23543 ms

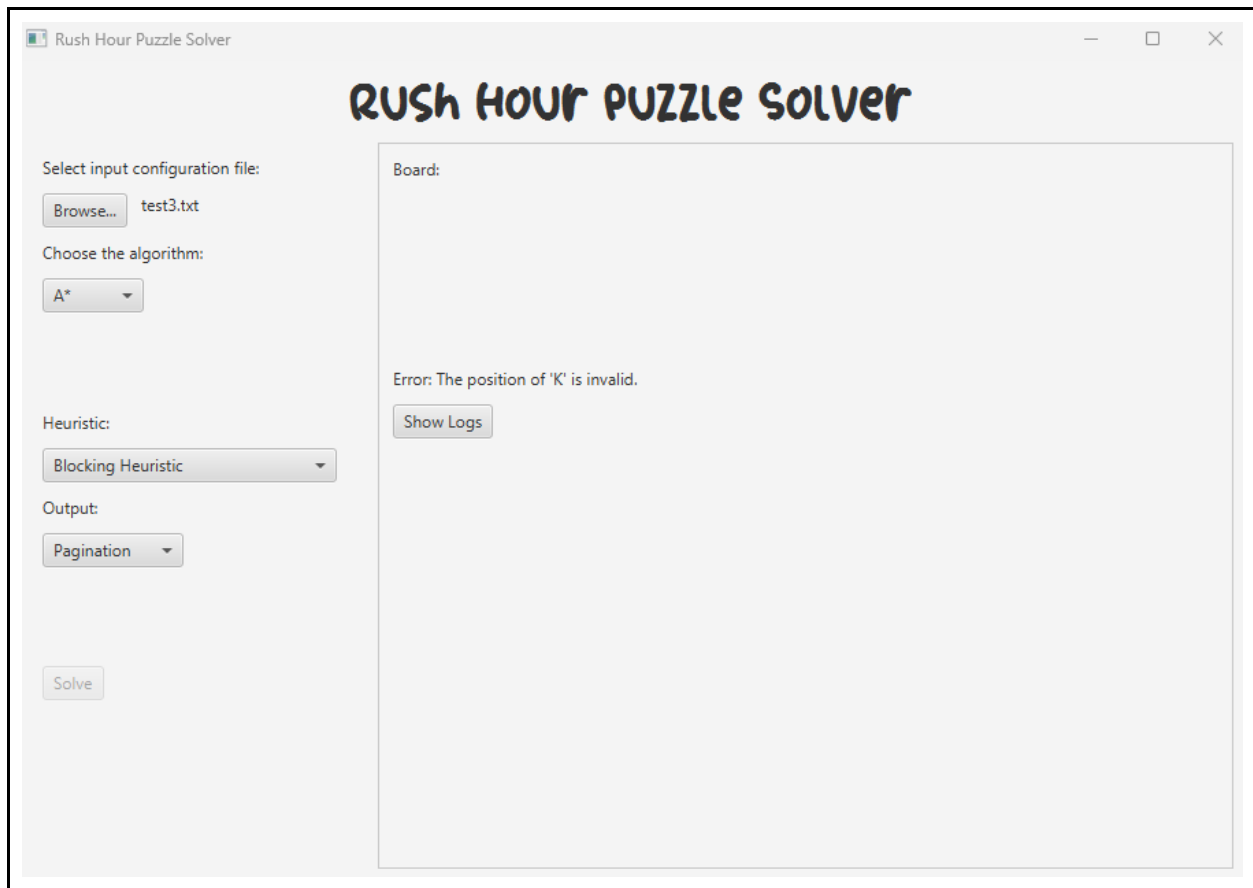
### 5.8.Test Case 8

Isi test3.txt

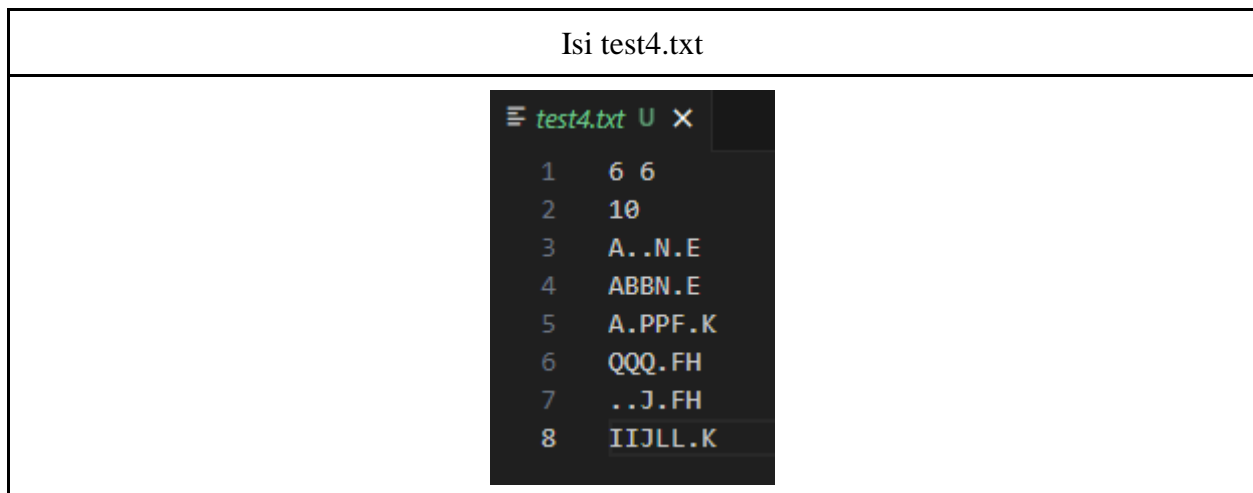


```
test3.txt U X
1 6 6
2 10
3 A..N.E
4 ABBN.E
5 A.PPF.
6 QQQ.FH
7 ..J.FH
8 IIJLL.
9      K
```

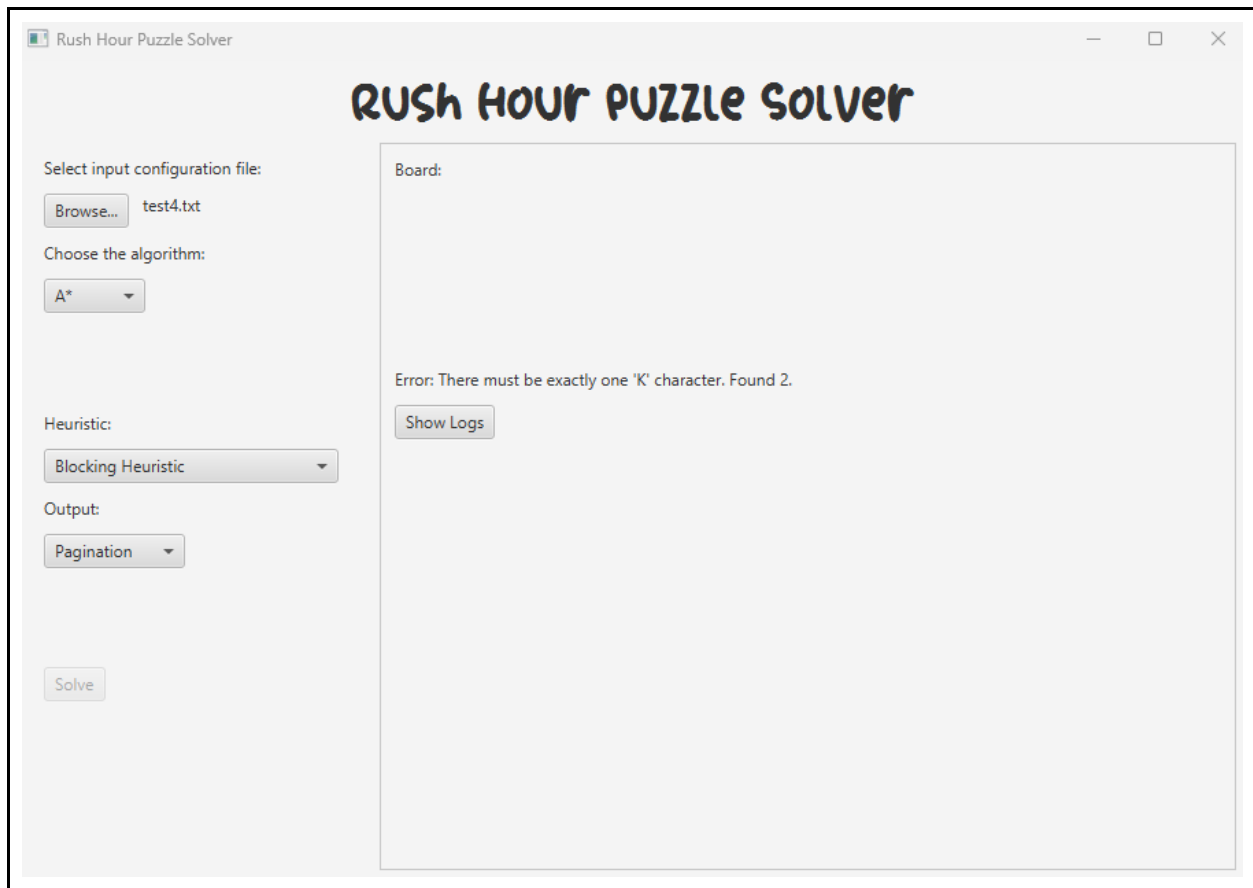
Tampilan pertama saat program baru dijalankan



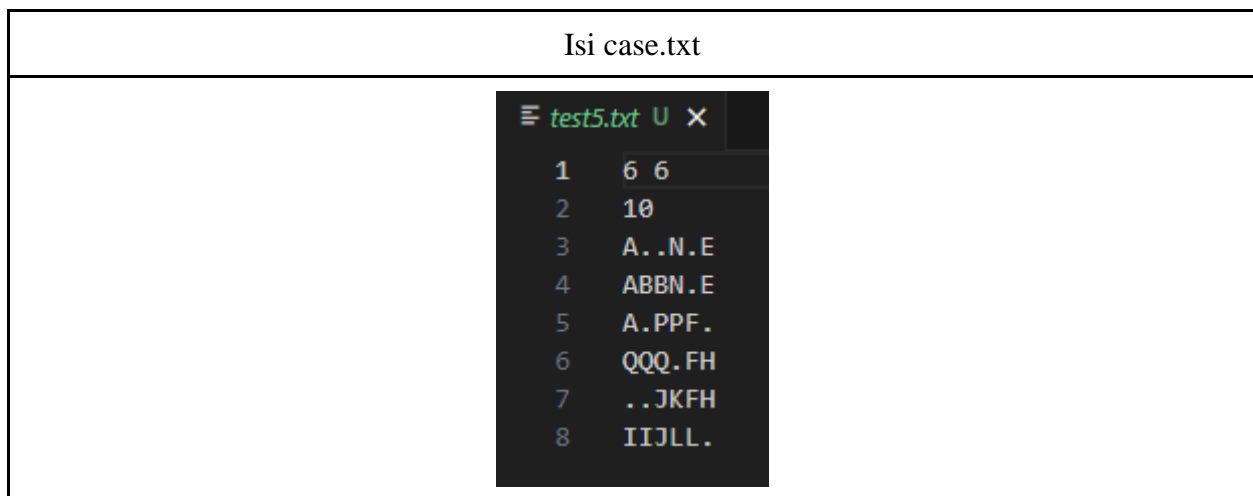
## 5.9. Test Case 9



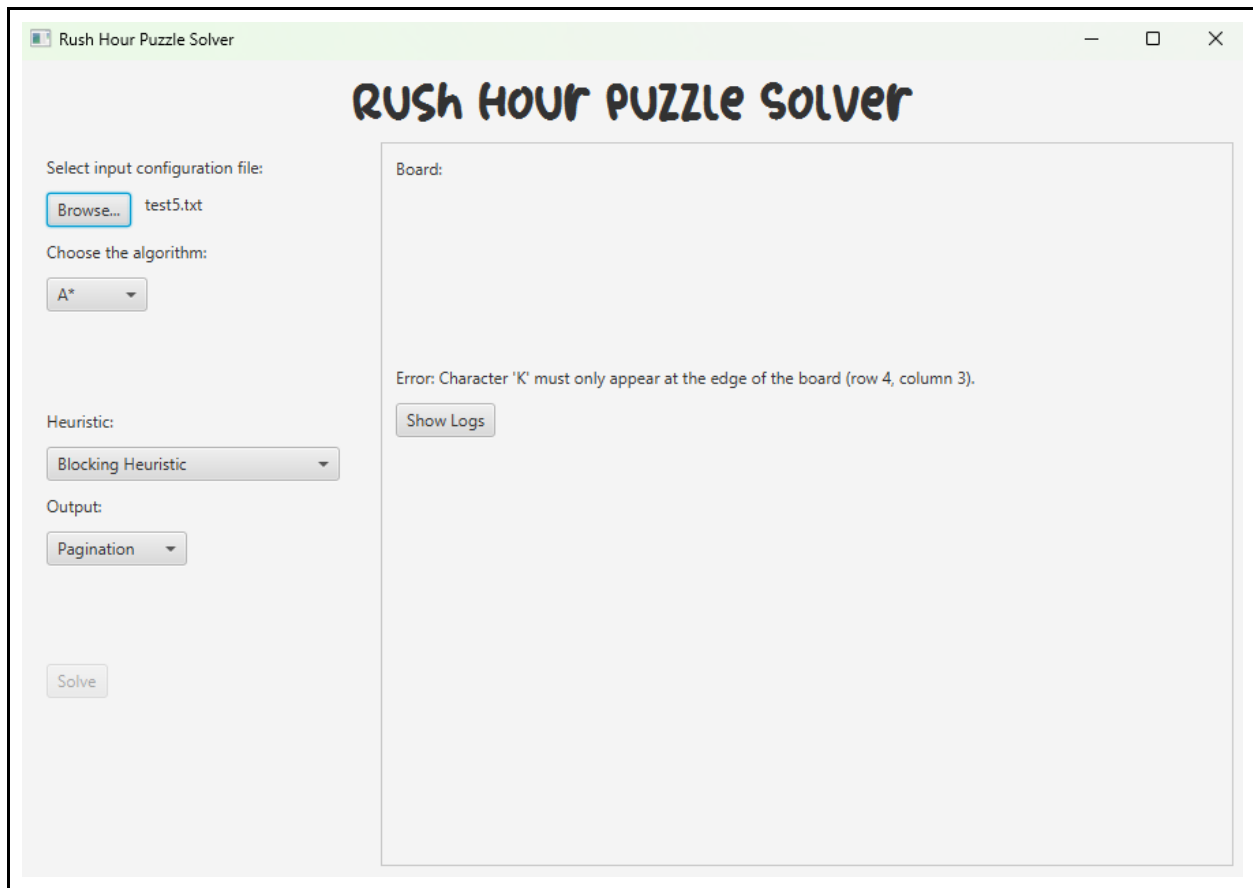
Tampilan pertama saat program baru dijalankan



## 5.10. Test Case 10



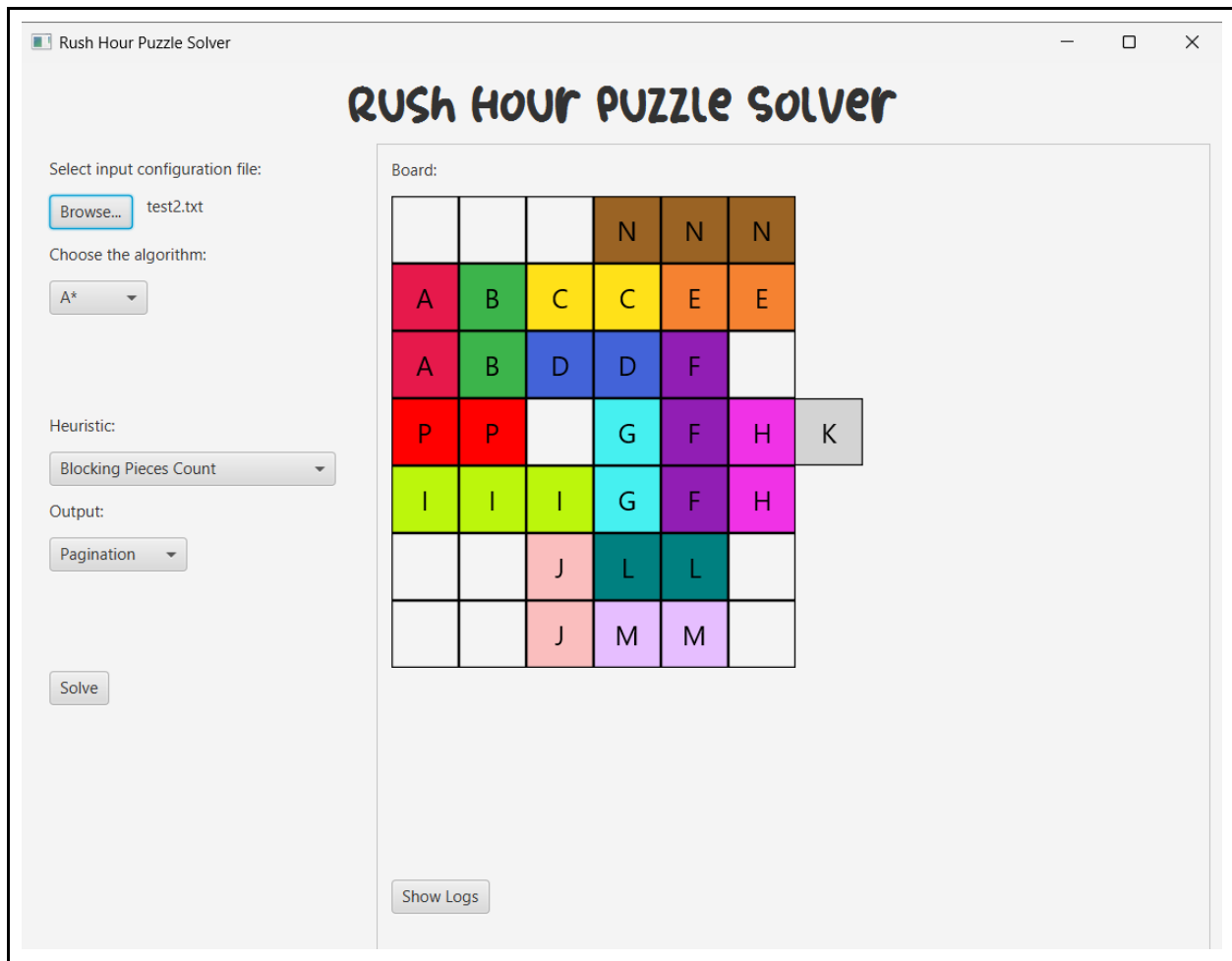
Tampilan pertama saat program baru dijalankan



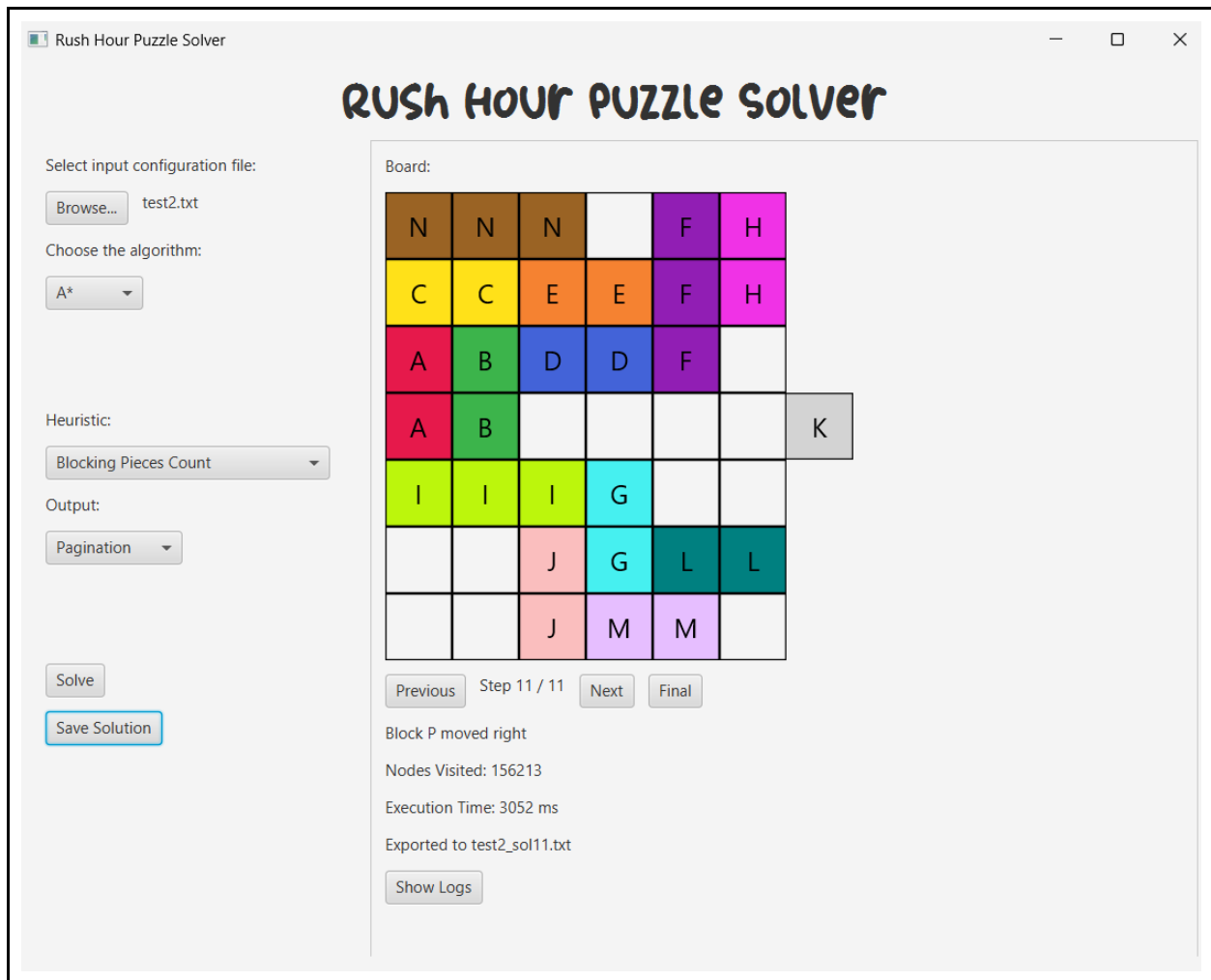
## 5.11. Test Case 11

Isi test2.txt	
test >	test2.txt
1	7 6
2	13
3	...NNN
4	ABCCEE
5	ABDDF.
6	PP.GFHK
7	IIIGFH
8	..JLL.
9	..JMM.

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test2.txt pada GUI*



*Output penyelesaian program test2.txt yang telah disimpan pada file test2\_sol11.txt*

Initial state

Step 0:

...NNN

ABCCEE

ABDDF.

PP.GFHK

IIIGFH

...JLL.

...JMM.

.

.

.

Step 11:

Block P moved right

NNN.FH

CCEEFH

ABDDF.

AB....K

IIIG..

..JGLL

..JMM.

Nodes Visited: 156213

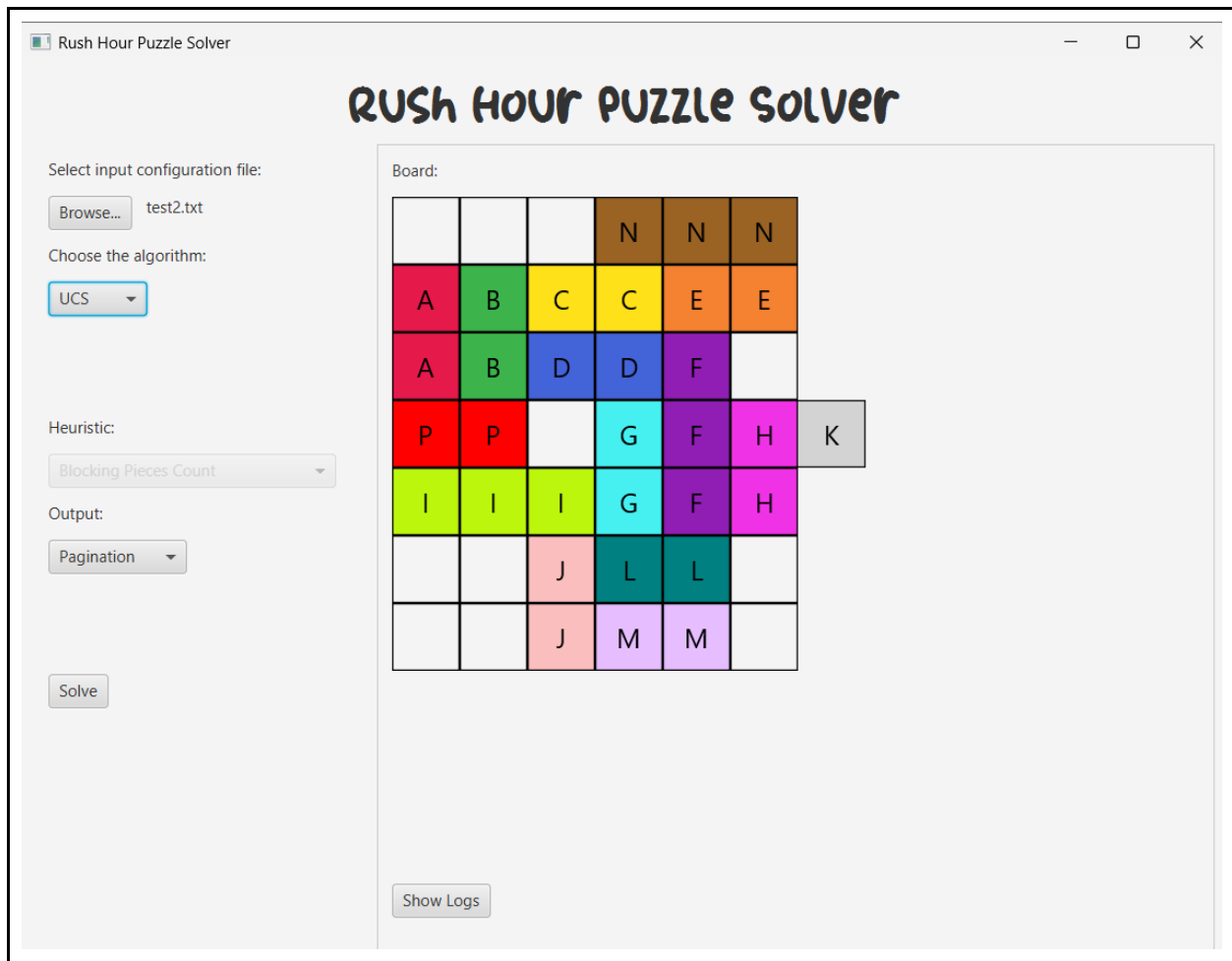
Execution Time: 3052 ms

### 5.12. Test Case 12

Isi test2.txt

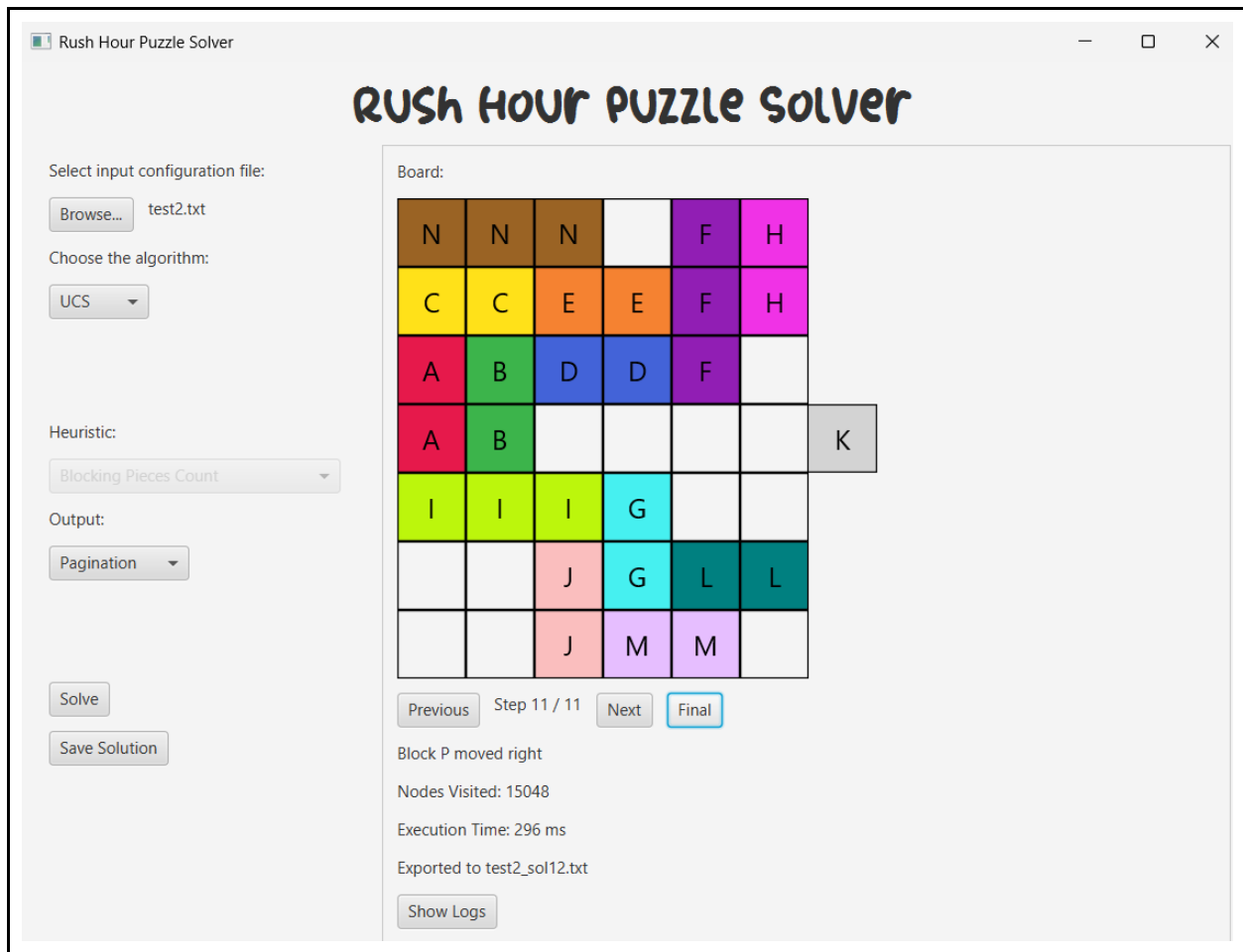
```
test > test2.txt
1 7 6
2 13
3 ...NNN
4 ABCCEE
5 ABDDF.
6 PP.GFHK
7 IIIGFH
8 ..JLL.
9 ..JMM.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test2.txt pada GUI*





*Output penyelesaian program test2.txt yang telah disimpan pada file test2\_sol12.txt*

Initial state

Step 0:

...NNN

ABCC EE

ABDD F.

PP.GFHK

IIIGFH

...JLL.

...JMM.

.

Step 11:

Block P moved right

NNN.FH

CCEEFH  
ABDDF.  
AB....K  
IIIG..  
..JGLL  
..JMM.

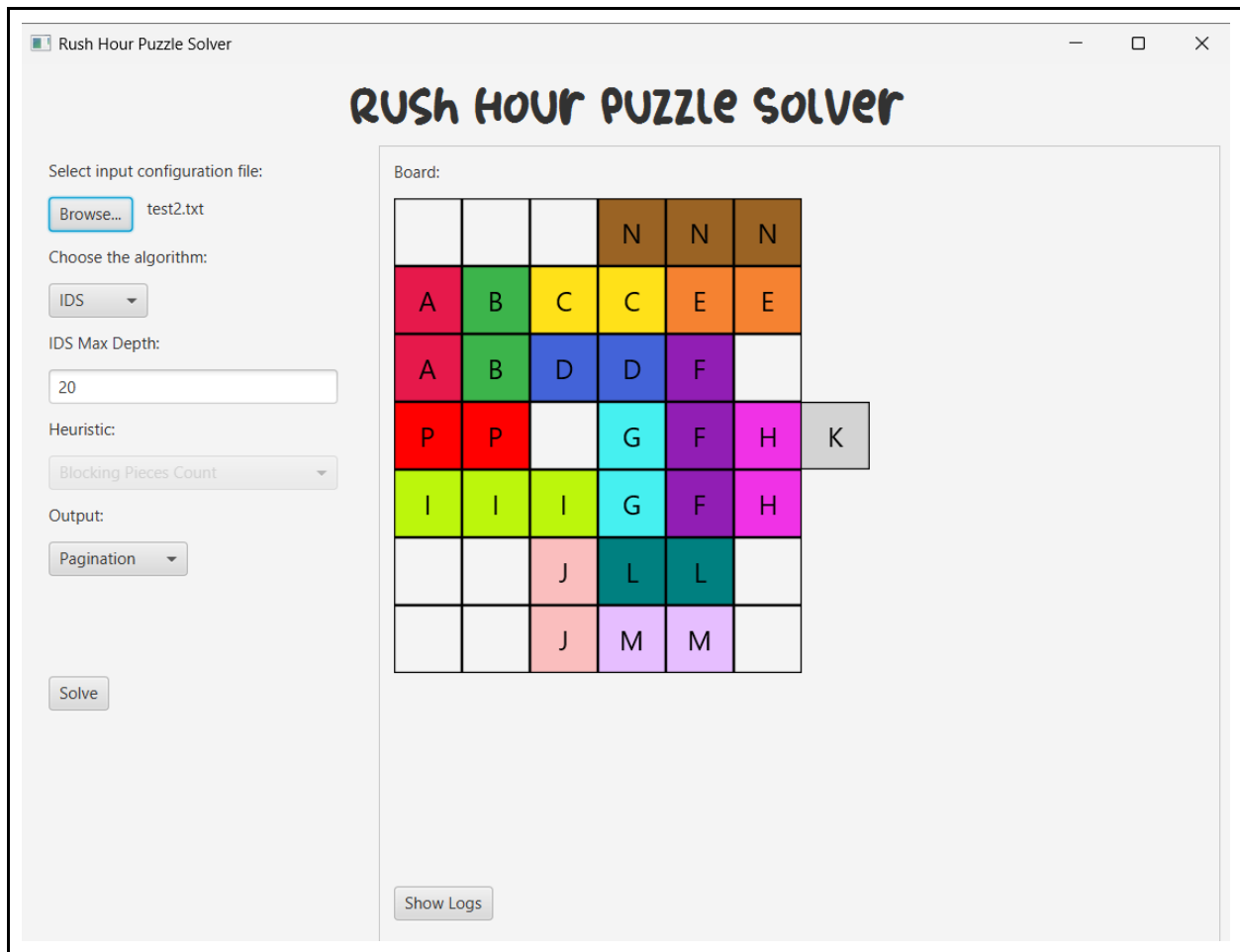
Nodes Visited: 15048  
Execution Time: 296 ms

### 5.13. Test Case 13

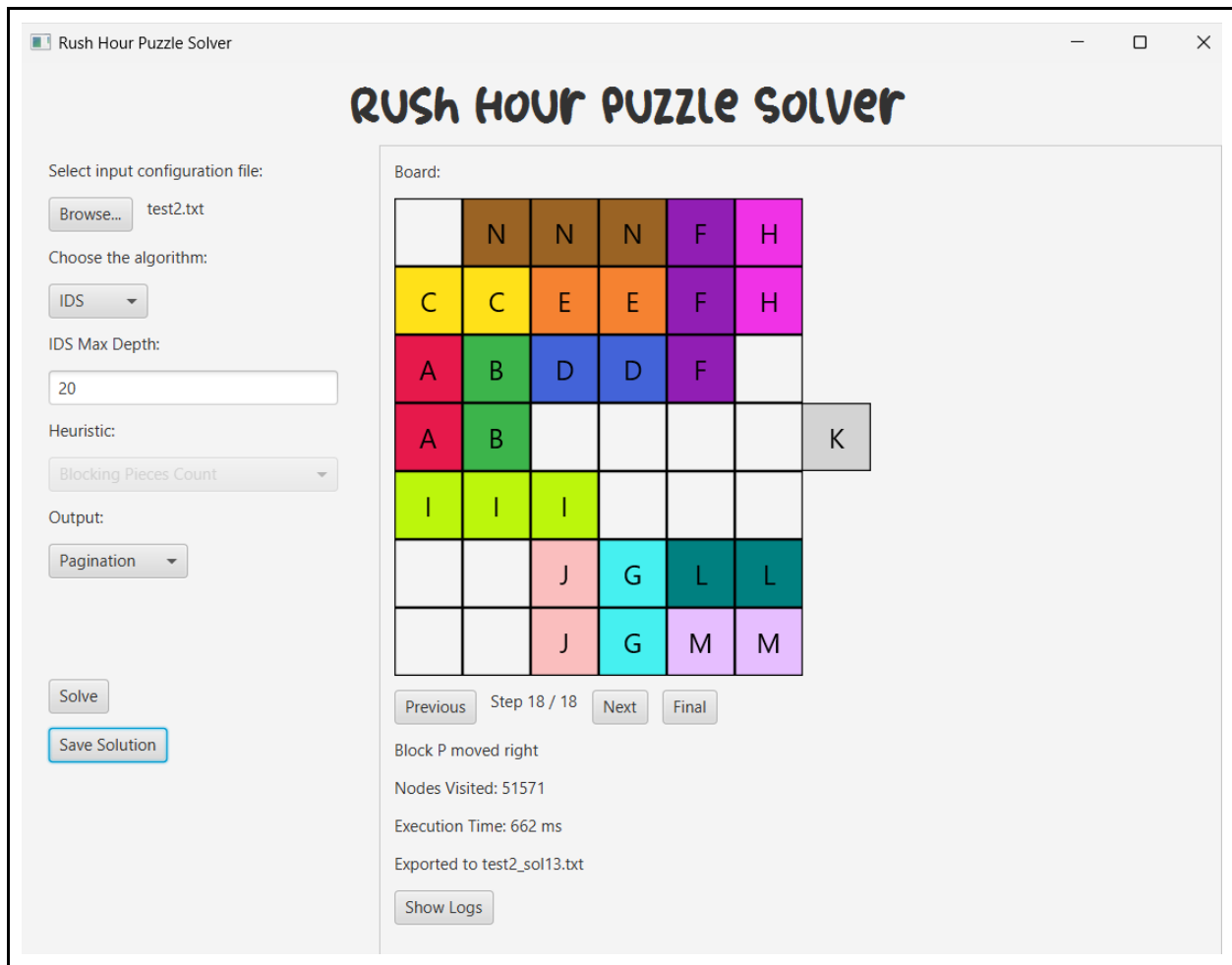
Isi test2.txt

```
test > test2.txt
1 7 6
2 13
3 ..NNN
4 ABCCEE
5 ABDDF.
6 PP.GFHK
7 IIIGFH
8 ..JLL.
9 ..JMM.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test2.txt pada GUI*



*Output penyelesaian program test2.txt yang telah disimpan pada file test2\_sol13.txt*

Initial state

Step 0:

...NNN

ABCC EE

ABDD F.

PP.GFHK

IIIGFH

...JLL.

...JMM.

.

.

.

Step 18:

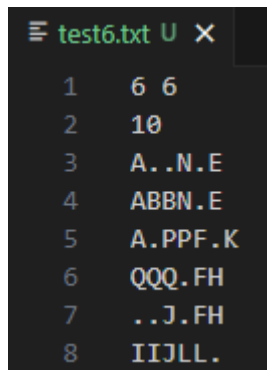
Block P moved right

.NNNFH  
CCEEFH  
ABDDF.  
AB....K  
III...  
..JGLL  
..JGMM

Nodes Visited: 51571  
Execution Time: 662 ms

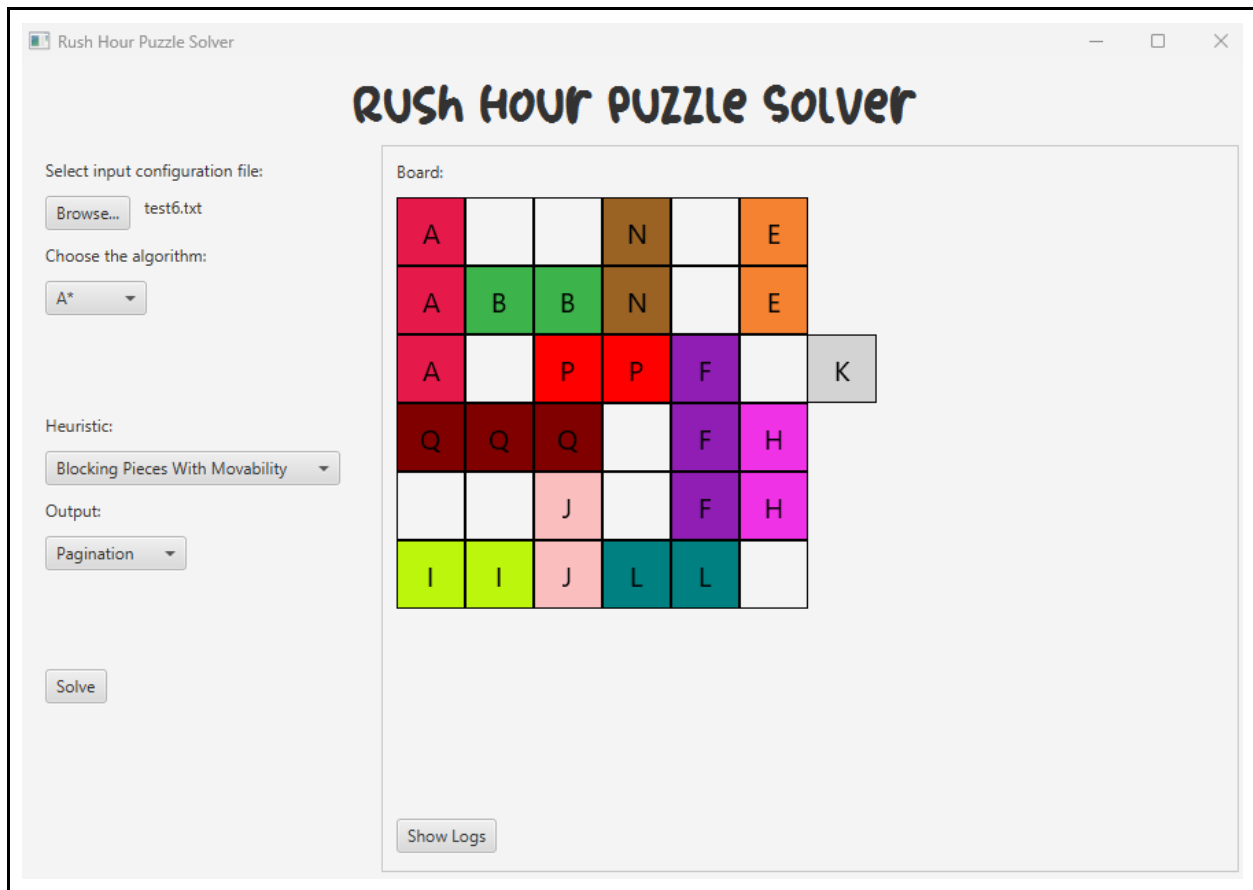
#### 5.14. Test Case 14

Isi test6.txt

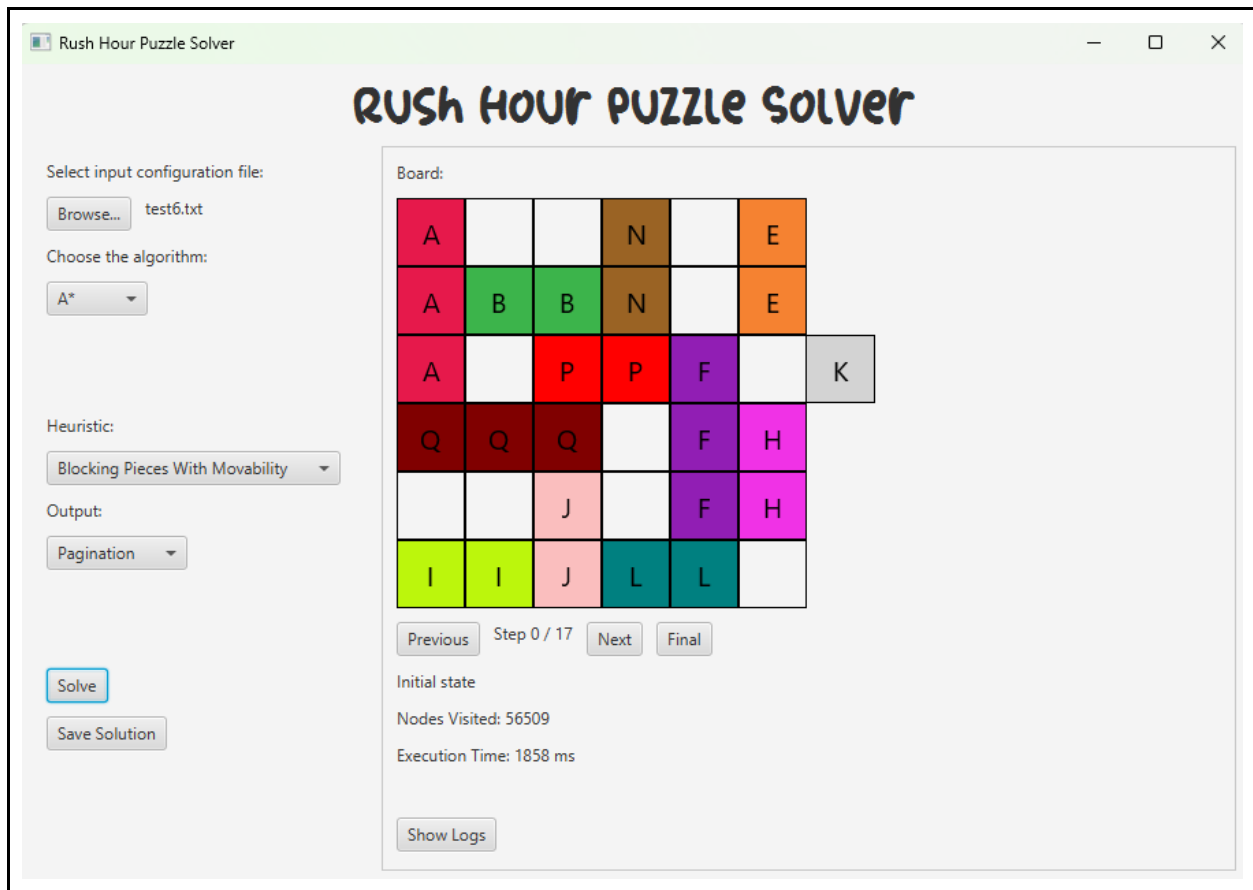


```
test6.txt U X
1 6 6
2 10
3 A..N.E
4 ABBN.E
5 A.PPF.K
6 QQQ.FH
7 ..J.FH
8 IIJLL.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test6.txt pada GUI*



*Output penyelesaian program test6.txt yang telah disimpan pada file test6\_sol14.txt*

Initial state

Step 0:

A..N.E

ABBN.E

A.PPF.K

QQQ.FH

..J.FH

IIJLL.

.

.

.

Step 17:

Block P moved right

..JN.E

BBJN.E

A.....K

AQQQF.

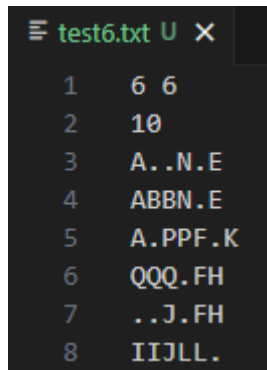
A...FH

IILLFH

Nodes Visited: 56509  
Execution Time: 1858 ms

### 5.15. Test Case 15

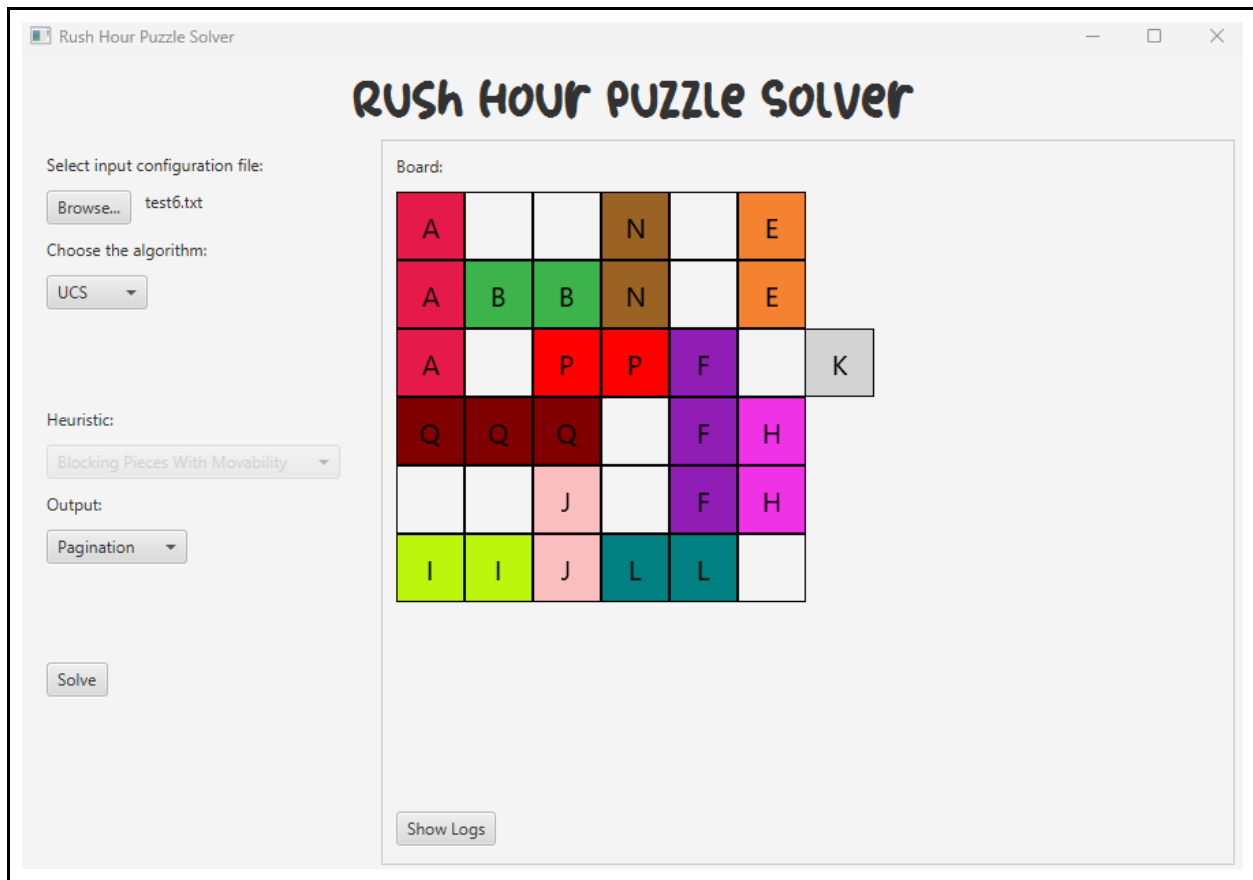
Isi test6.txt



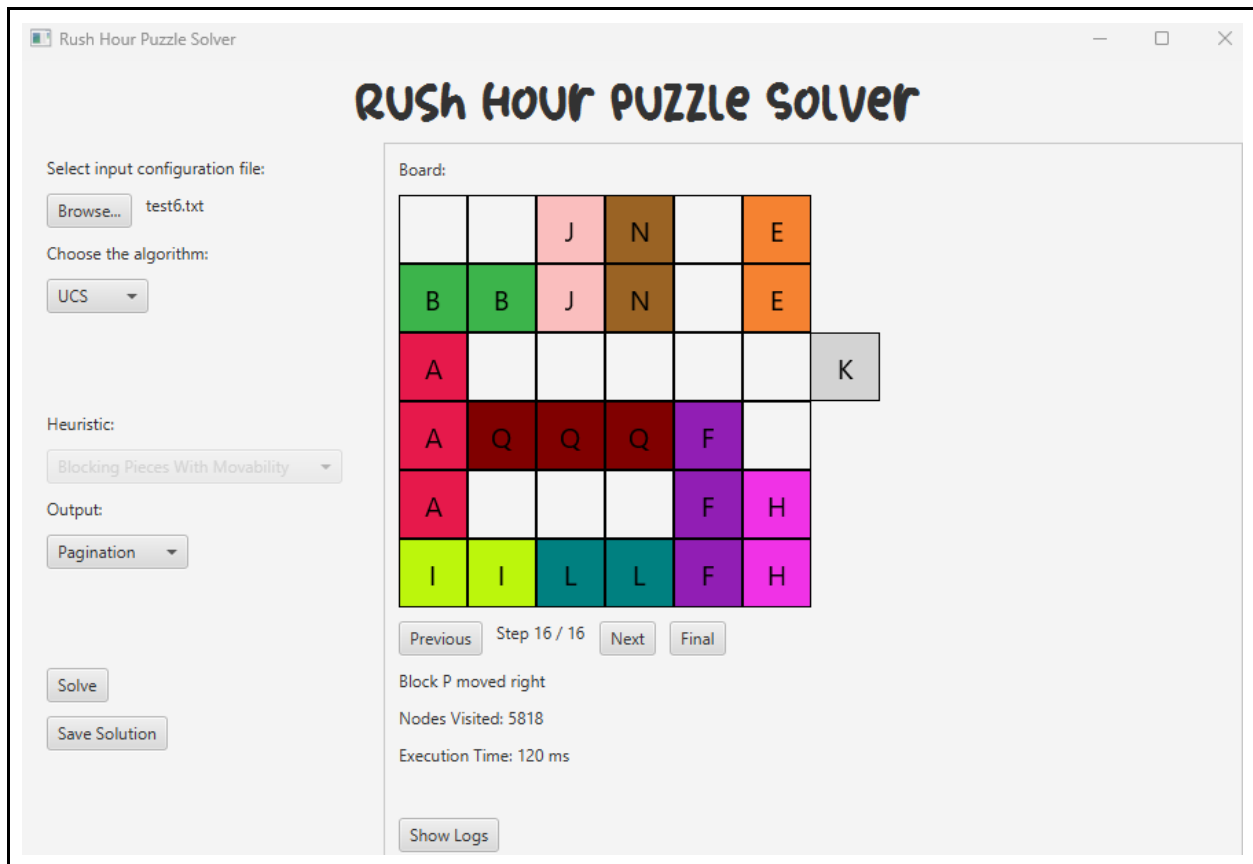
```
test6.txt U X
1 6 6
2 10
3 A..N.E
4 ABBN.E
5 A.PPF.K
6 QQQ.FH
7 ..J.FH
8 IIJLL.
```

Tampilan pertama saat program baru dijalankan





*Output penyelesaian program test6.txt pada GUI*



*Output penyelesaian program test6.txt yang telah disimpan pada file test6\_sol15.txt*

Initial state

Step 0:

A..N.E

ABBN.E

A.PPF.K

QQQ.FH

...J.FH

IIJLL.

.

.

.

Step 16:

Block P moved right

...JN.E

BBJN.E

A.....K

AQQQF.

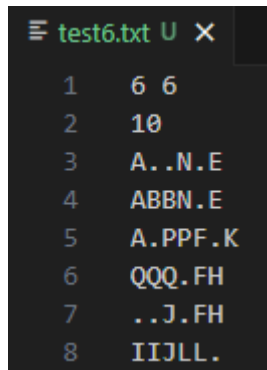
A...FH

IILLFH

Nodes Visited: 5818  
Execution Time: 120 ms

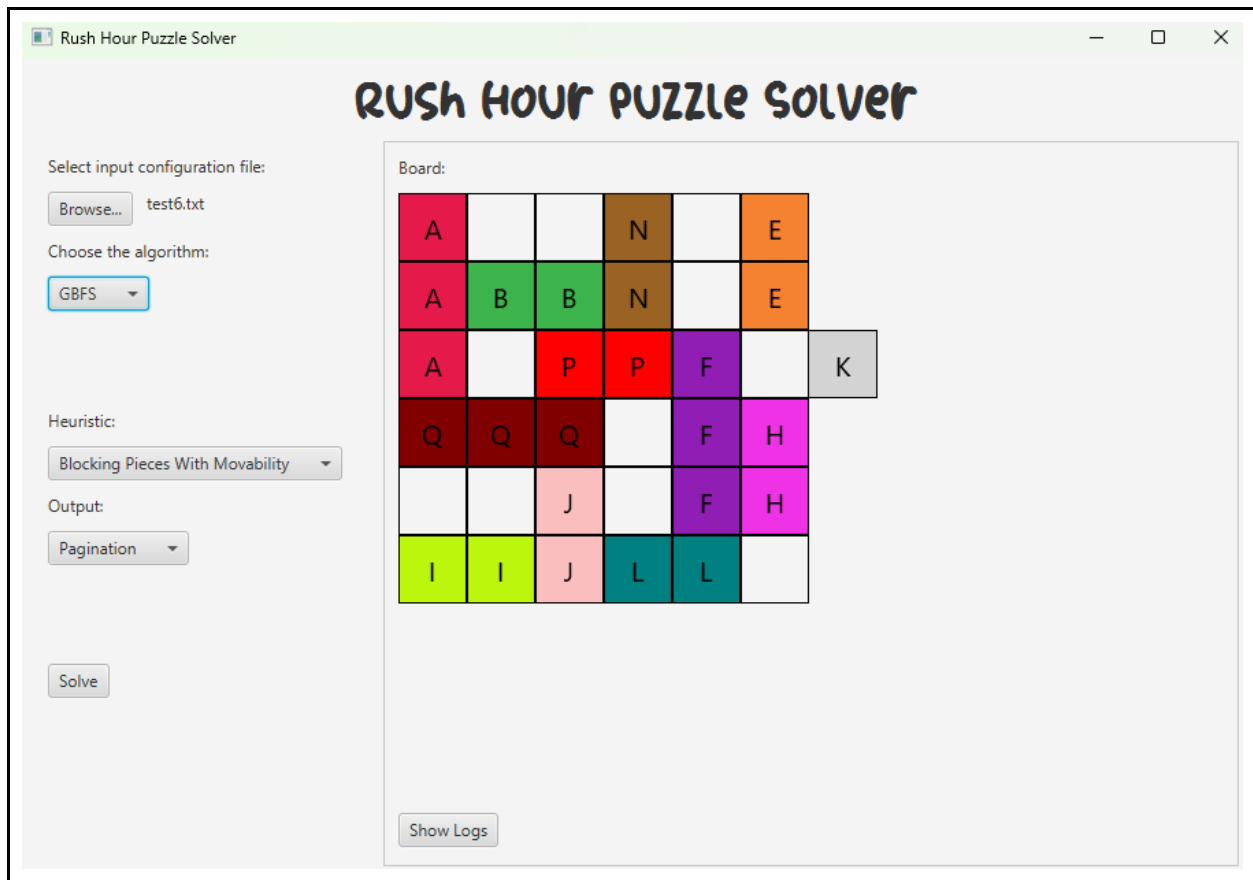
### 5.16. Test Case 16

Isi test6.txt

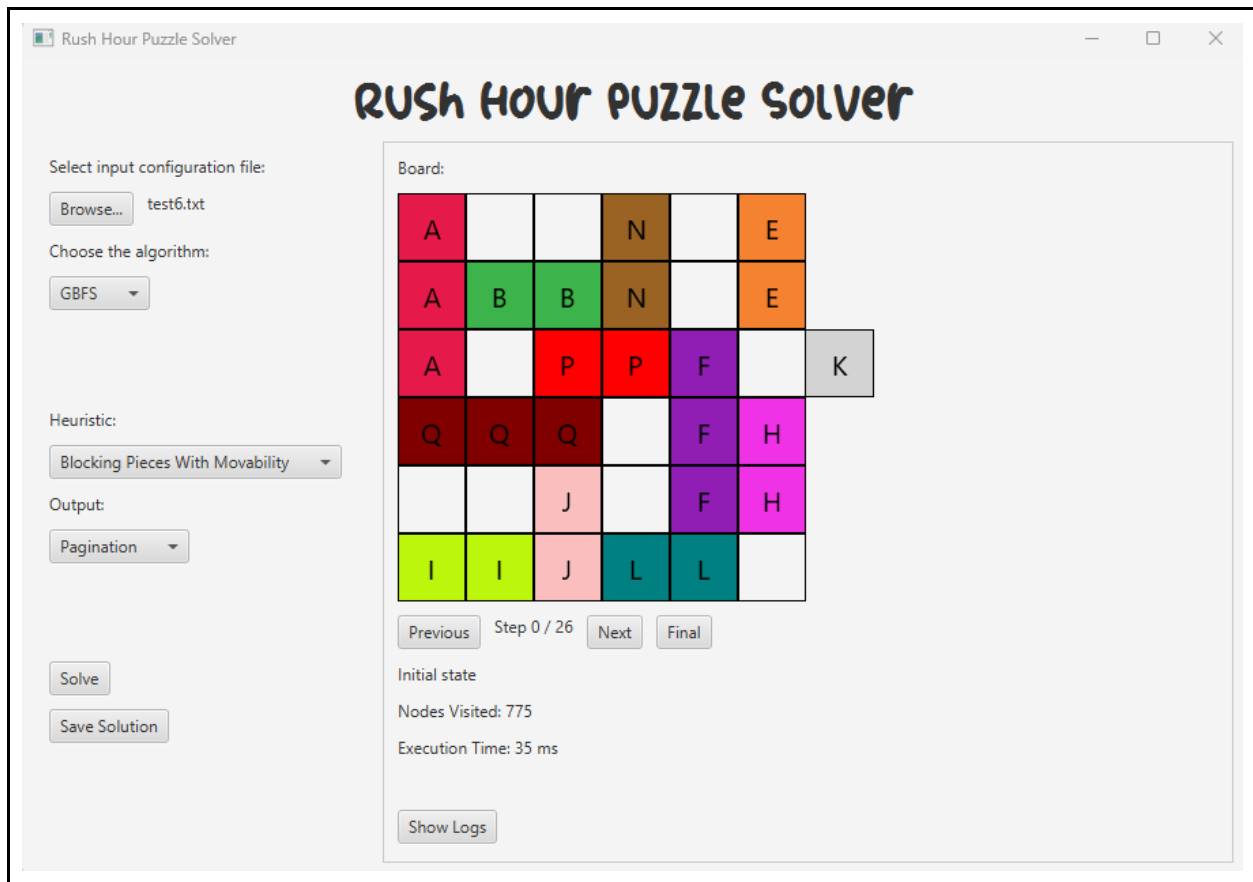


```
1 6 6
2 10
3 A..N.E
4 ABBN.E
5 A.PPF.K
6 QQQ.FH
7 ..J.FH
8 IIJLL.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test6.txt pada GUI*



*Output penyelesaian program test6.txt yang telah disimpan pada file test6\_sol16.txt*

Initial state

Step 0:

A..N.E

ABBN.E

A.PPF.K

QQQ.FH

..J.FH

IIJLL.

.

.

.

Step 26:

Block P moved right

...JN.E

BBJN.E

A.....K

AQQQF.

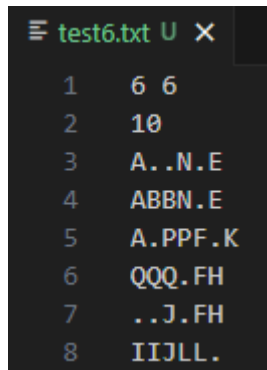
A...FH

IILLFH

Nodes Visited: 775  
Execution Time: 35 ms

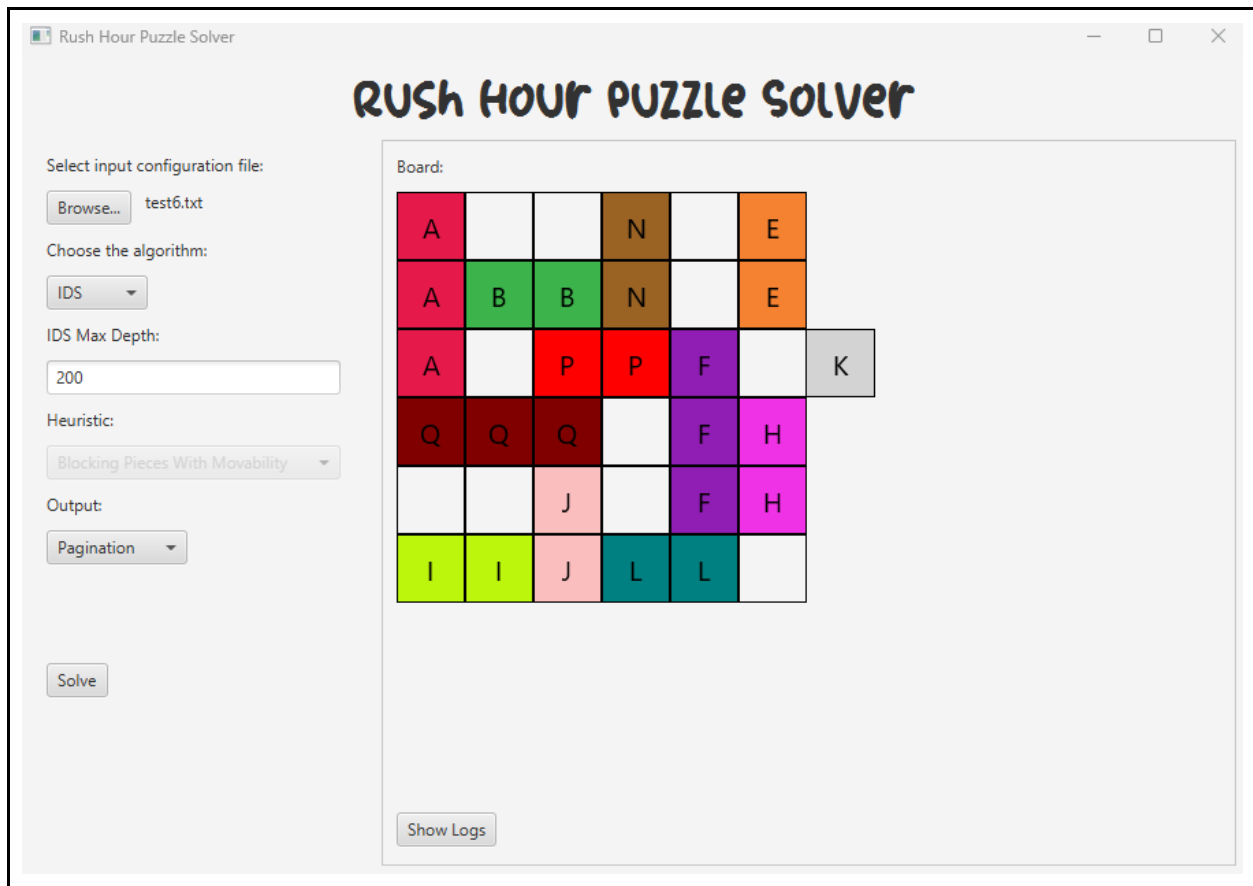
### 5.17. Test Case 17

Isi test6.txt

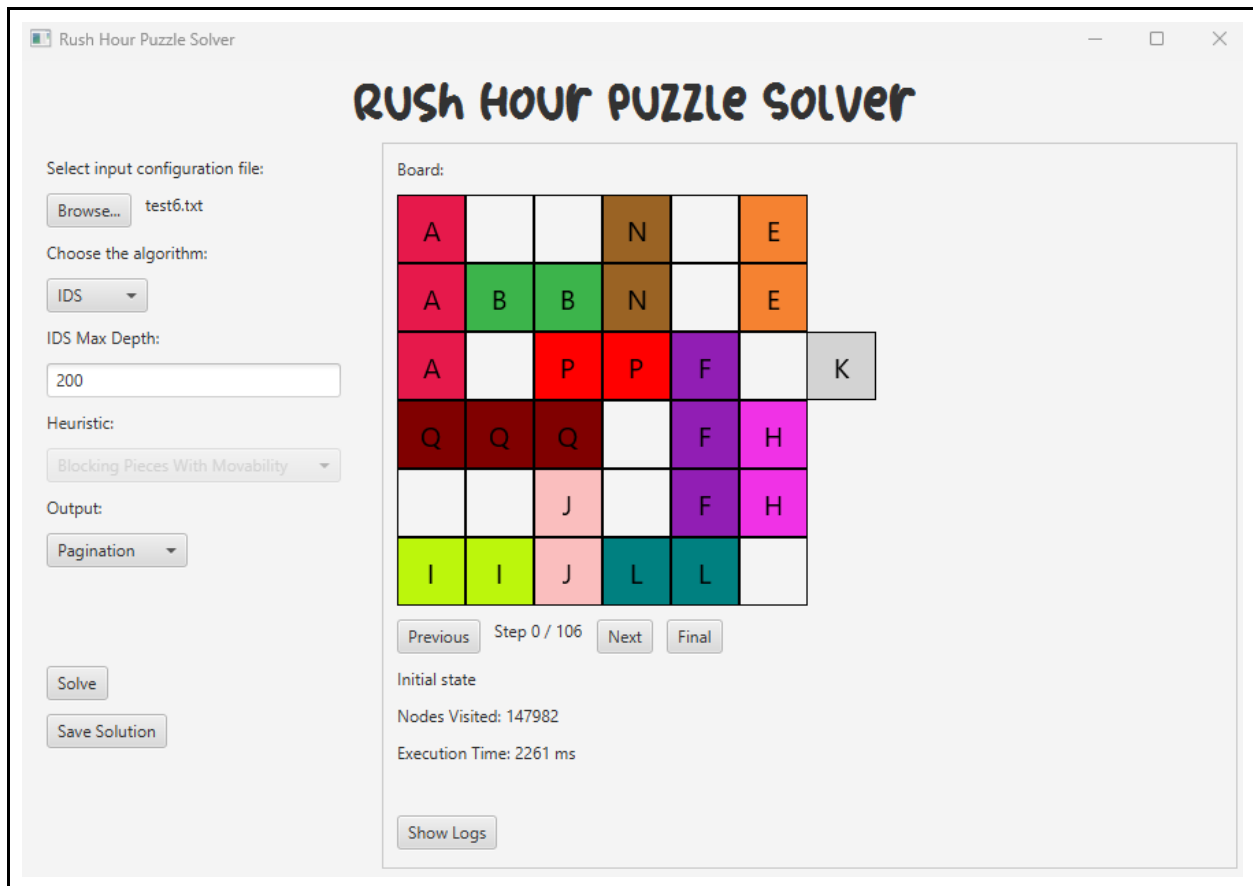


```
test6.txt U X
1 6 6
2 10
3 A..N.E
4 ABBN.E
5 A.PPF.K
6 QQQ.FH
7 ..J.FH
8 IIJLL.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test6.txt pada GUI*



*Output penyelesaian program test6.txt yang telah disimpan pada file test6\_sol17.txt*

Initial state

Step 0:

A..N.E

ABBN.E

A.PPF.K

QQQ.FH

..J.FH

IJLL.

.

.

.

Step 106:

Block P moved right

..JN.E

BBJN.E

A.....K

AQQQF.

A...FH

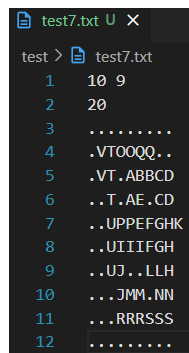
IILLFH



Nodes Visited: 147982  
Execution Time: 2261 ms

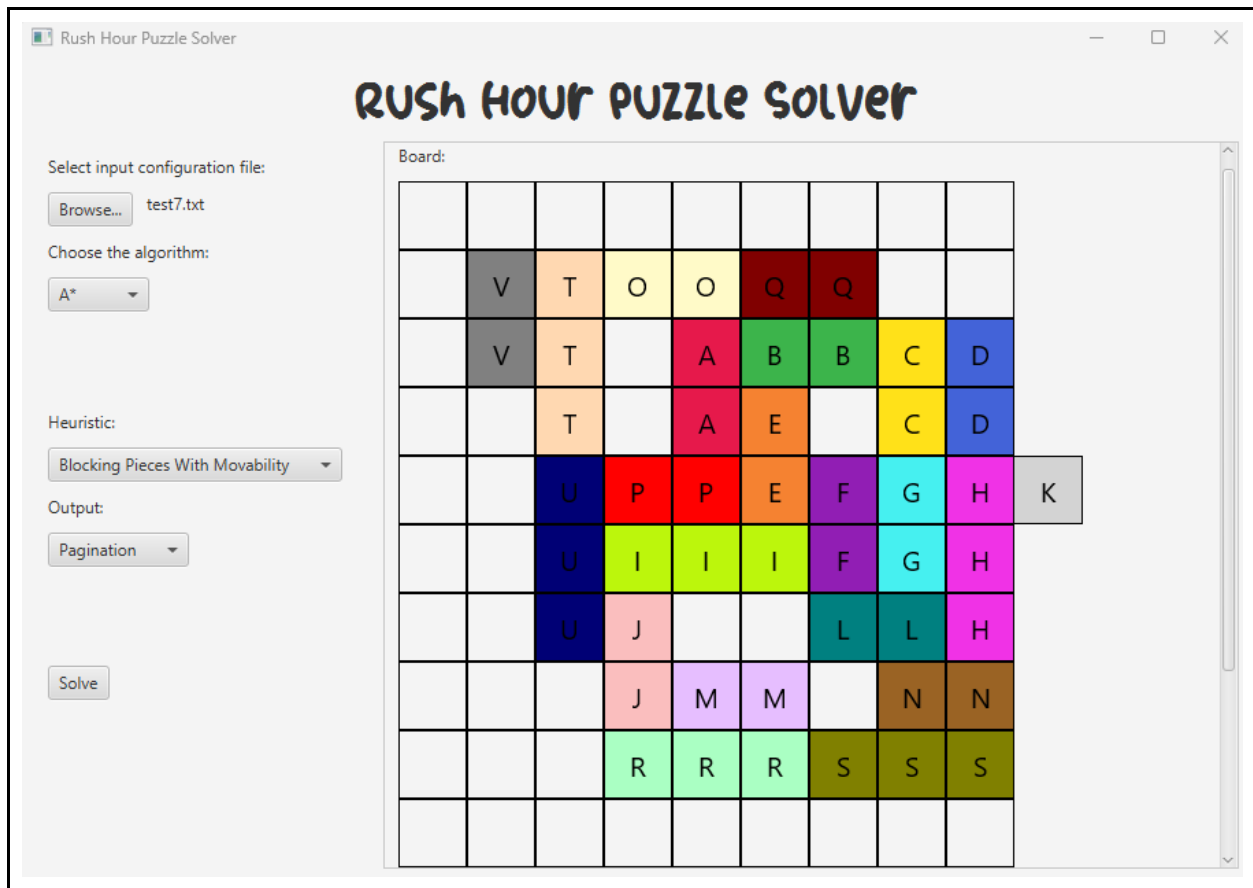
### 5.18. Test Case 18

Isi test7.txt

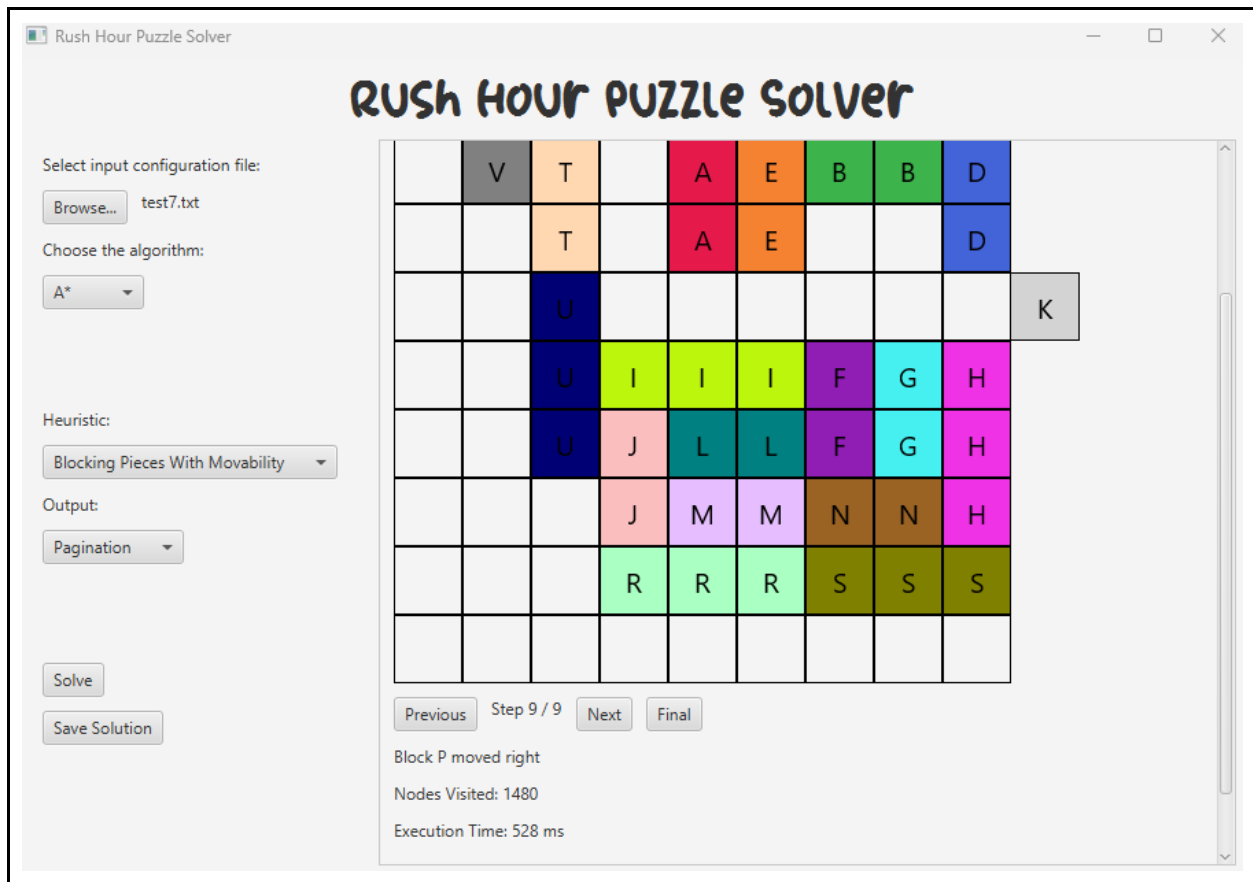


```
test7.txt U X
test > test7.txt
1 10 9
2 20
3 .....
4 .VT00QQ..
5 .VT.ABBCD
6 ..T.AE.CD
7 ..UPPEFGHK
8 ..UIIIFGH
9 ..UJ..LLH
10 ...JMM.NN
11 ...RRRSSS
12 .....
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test7.txt pada GUI*



*Output penyelesaian program test7.txt yang telah disimpan pada file test7\_sol18.txt*

Initial state

Step 0:

```

.....
.VTOOQQ..
.VT.ABBCD
..T.AE.CD
..UPPEFGHK
..UIIIFGH
..UJ..LLH
...JMM.NN
...RRRSSS
.....
.
.
.

```

Step 9:

Block P moved right

```

.....C.
.VTOOQQC.

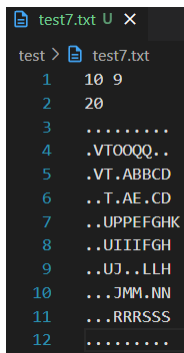
```

```
.VT.AEBBD
..T.AE..D
..U.....K
..UIIFGH
..UJLLFGH
...JMMNNH
...RRRSSS
.....
```

Nodes Visited: 1480  
Execution Time: 528 ms

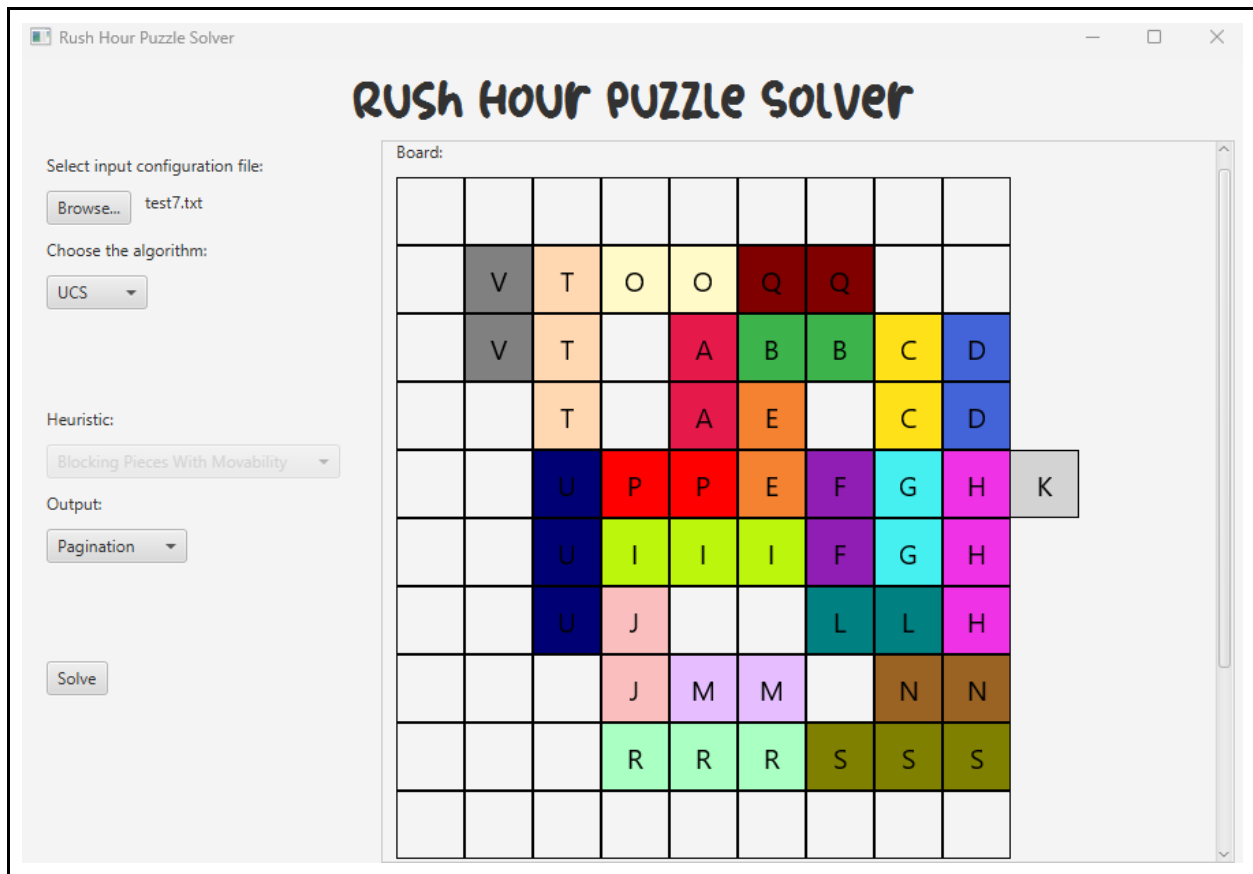
### 5.19. Test Case 19

Isi test7.txt

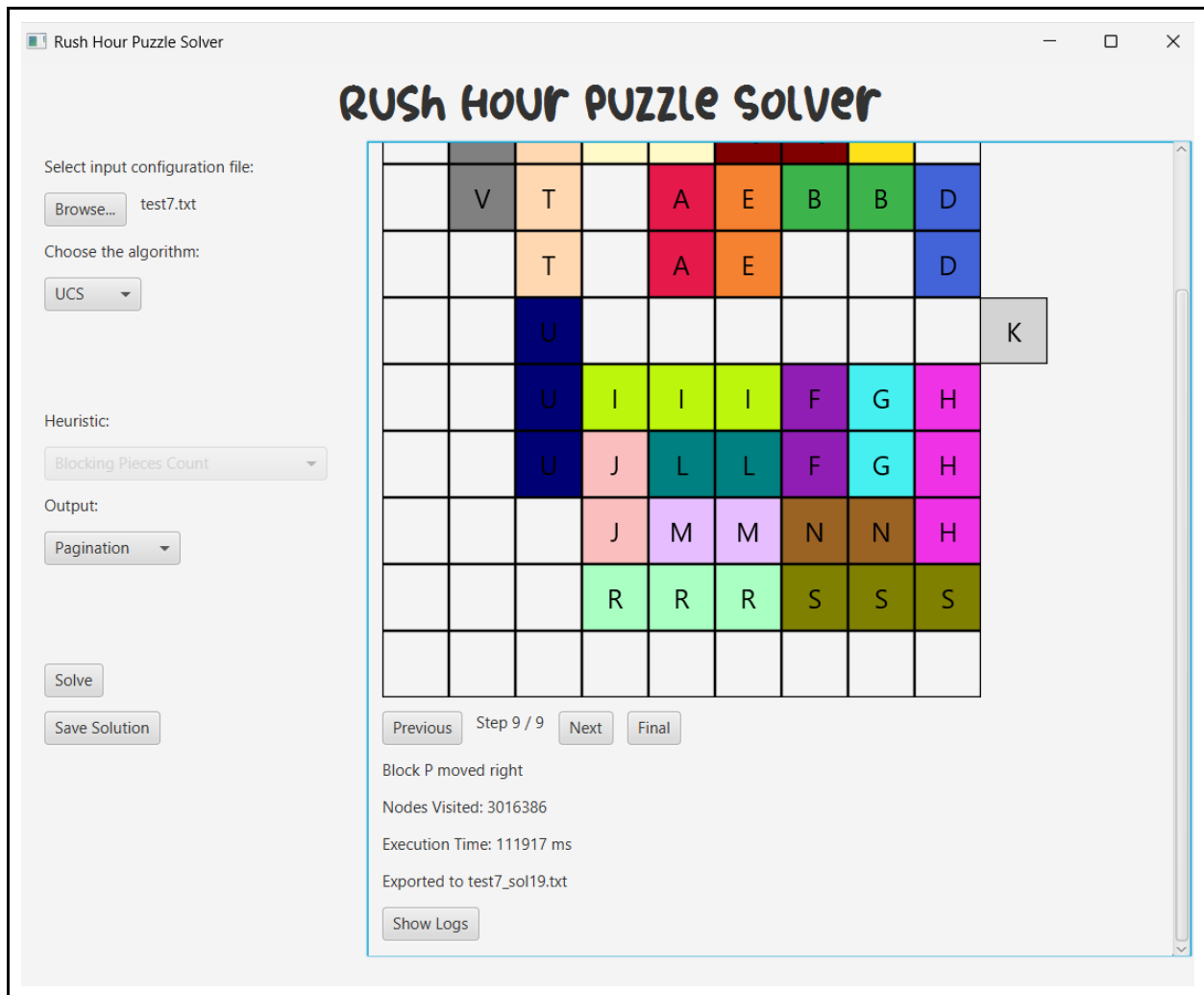


```
test > test7.txt
1 10 9
2 20
3 .....
4 .VTOOQQ..
5 .VT.ABBBD
6 ..T.AE.CD
7 ..UPPEFGHK
8 ..UIIFGH
9 ..UJ..LLH
10 ...JMM.NN
11 ...RRRSSS
12 .....
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test7.txt pada GUI*



*Output penyelesaian program test7.txt yang telah disimpan pada file test7\_sol19.txt*

Initial state

Step 0:

```

.....
.VTOOQQ..
.VT.ABBCD
..T.AE.CD
..UPPEFGHK
..UIIFGH
..UJ..LLH
...JMM.NN
...RRRSS
.....

```

```

.
.

```

.

Step 9:

Block P moved right

.....C.

.VTOOQQC.

.VT.AEBBD

..T.AE..D

..U.....K

..UIIFGH

..UJLLFGH

...JMMNNH

...RRRSSS

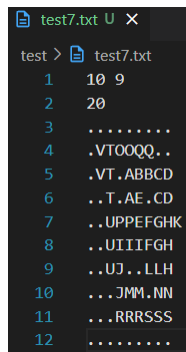
.....

Nodes Visited: 3016386

Execution Time: 111917 ms

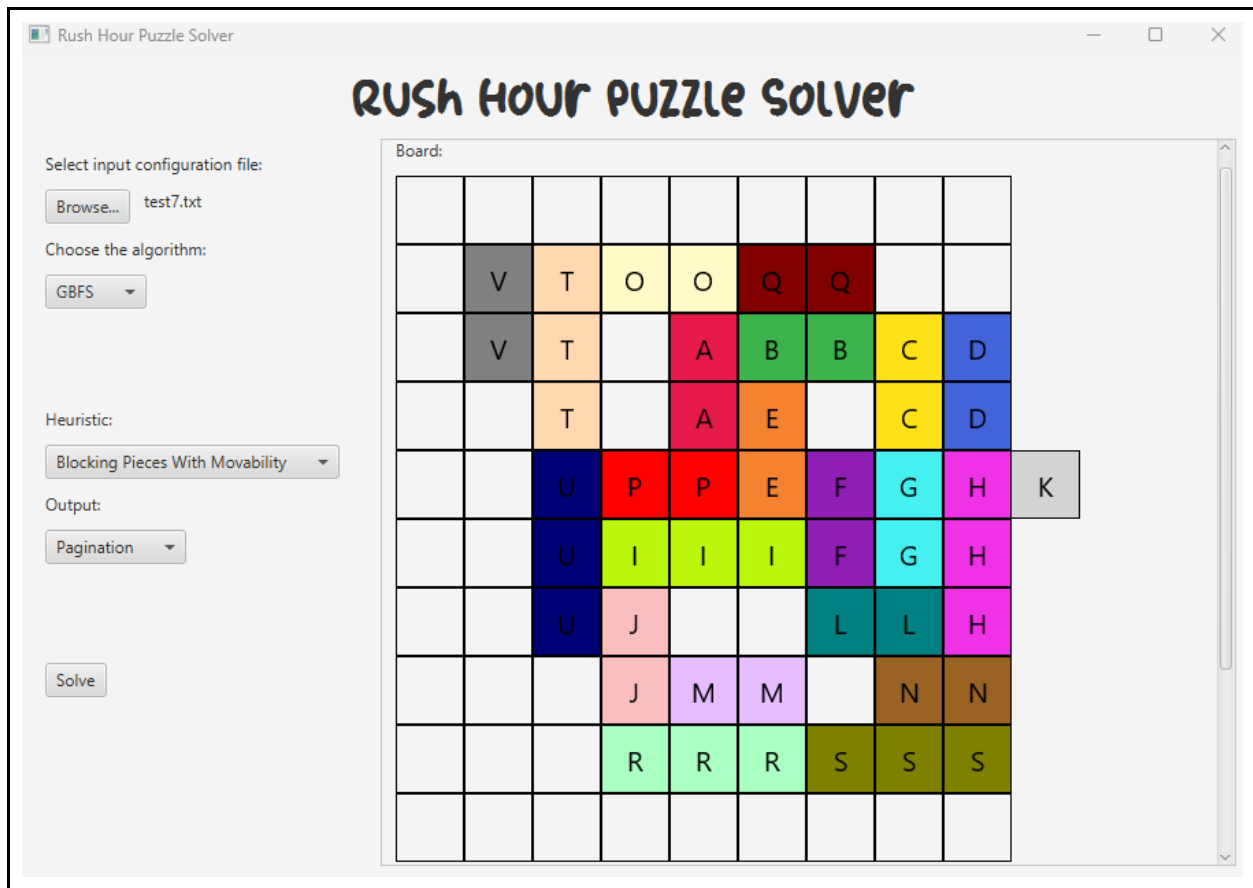
## 5.20. Test Case 20

Isi test7.txt



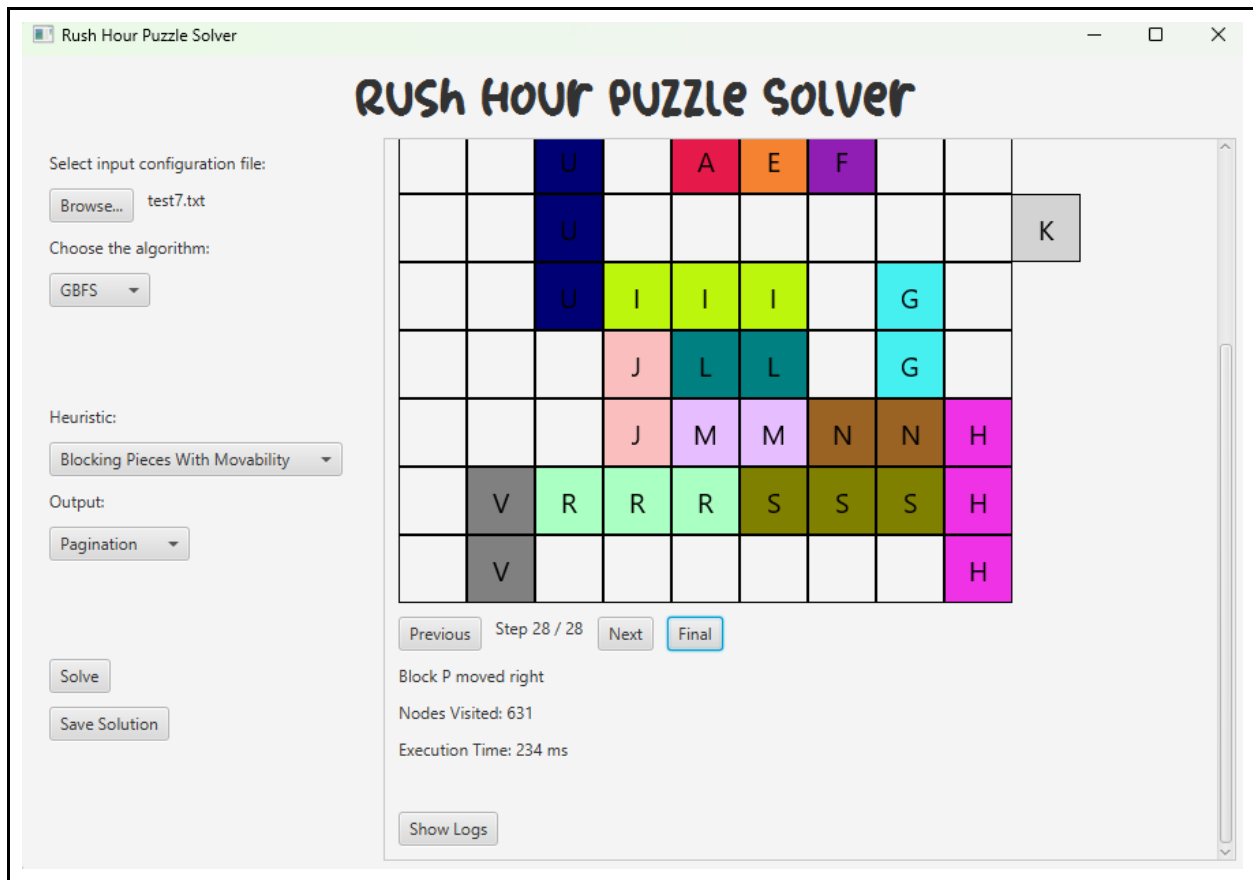
```
test7.txt U X
test > test7.txt
1 10 9
2 20
3 .....
4 .VTOOQQ.
5 .VT.ABBBD
6 ..T.AE.CD
7 ..UPPEFGHK
8 ..UIIFGH
9 ..UJ..LLH
10 ...JMM.NN
11 ...RRRSSS
12 .....
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test7.txt pada GUI*





*Output penyelesaian program test7.txt yang telah disimpan pada file test7\_sol20.txt*

Initial state

Step 0:

```

.....
.VTOOQQ..
.VT.ABBCD
..T.AE.CD
..UPPEFGHK
..UIIIFGH
..UJ..LLH
...JMM.NN
...RRRSSS
.....
.
.
.

```

Step 28:

Block P moved right

```

..T....CD
OOTQQ..CD

```

```
..T.AEFBB
..U.AEF..
..U.....K
..UIII.G.
...JLL.G.
...JMMNNH
.VRRRSSSH
.V.....H
```

Nodes Visited: 631  
Execution Time: 234 ms

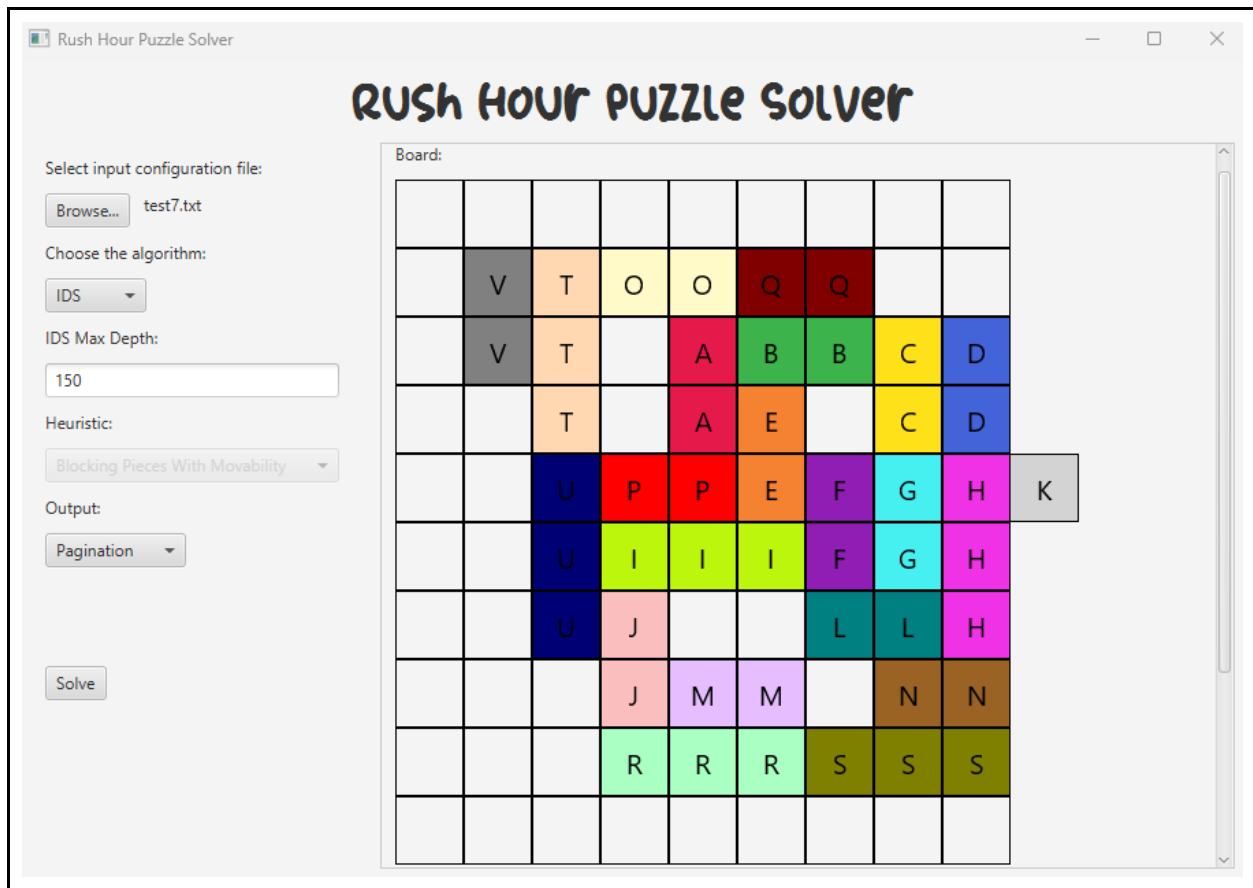
### 5.21. Test Case 21

Isi test7.txt

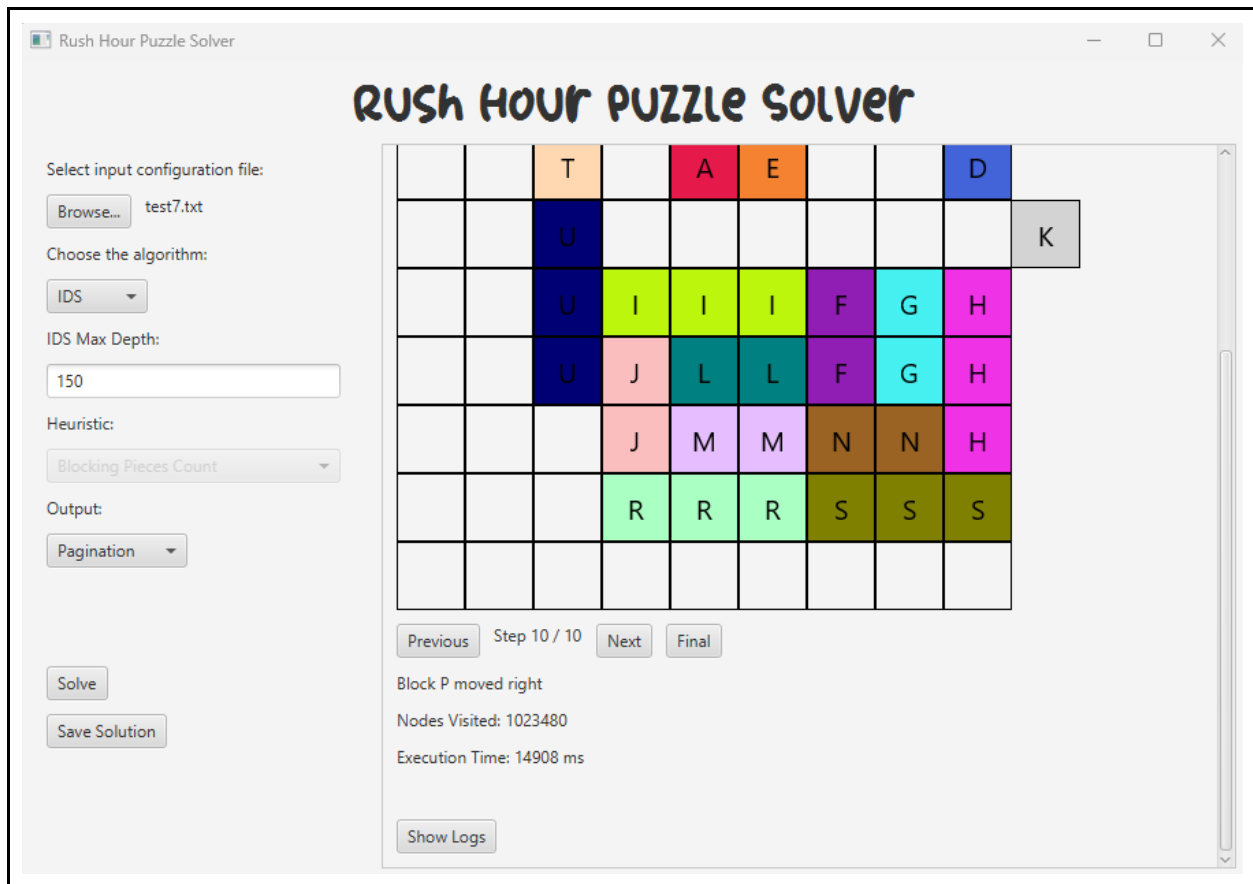


```
test > test7.txt
1 10 9
2 20
3 .....
4 .VTOOQQ.
5 .VT.ABB CD
6 ..T.AE.CD
7 ..UPPEFGHK
8 ..UIIIIFGH
9 ..UJ..LLH
10 ...JMM.NN
11 ...RRRSSS
12 .....
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test7.txt pada GUI*



*Output penyelesaian program test7.txt yang telah disimpan pada file test7\_sol21.txt*

Initial state

Step 0:

```

.....
.VTOOQQ..
.VT.ABBCD
..T.AE.CD
..UPPEFGHK
..UIIFGH
..UJ..LLH
...JMM.NN
...RRRSSS

```

.....

.  
.
  
.

Step 10:

Block P moved right

```

.....C.
.VTOOQQC.

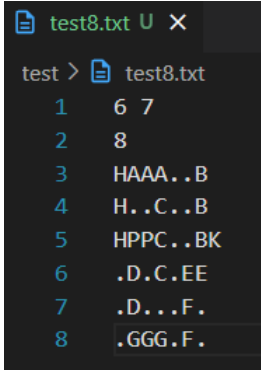
```

```
.VT.AEBBD
..T.AE..D
..U.....K
..UIIFGH
..UJLLFGH
...JMMNNH
...RRRSSS
.....
```

Nodes Visited: 1023480  
Execution Time: 14908 ms

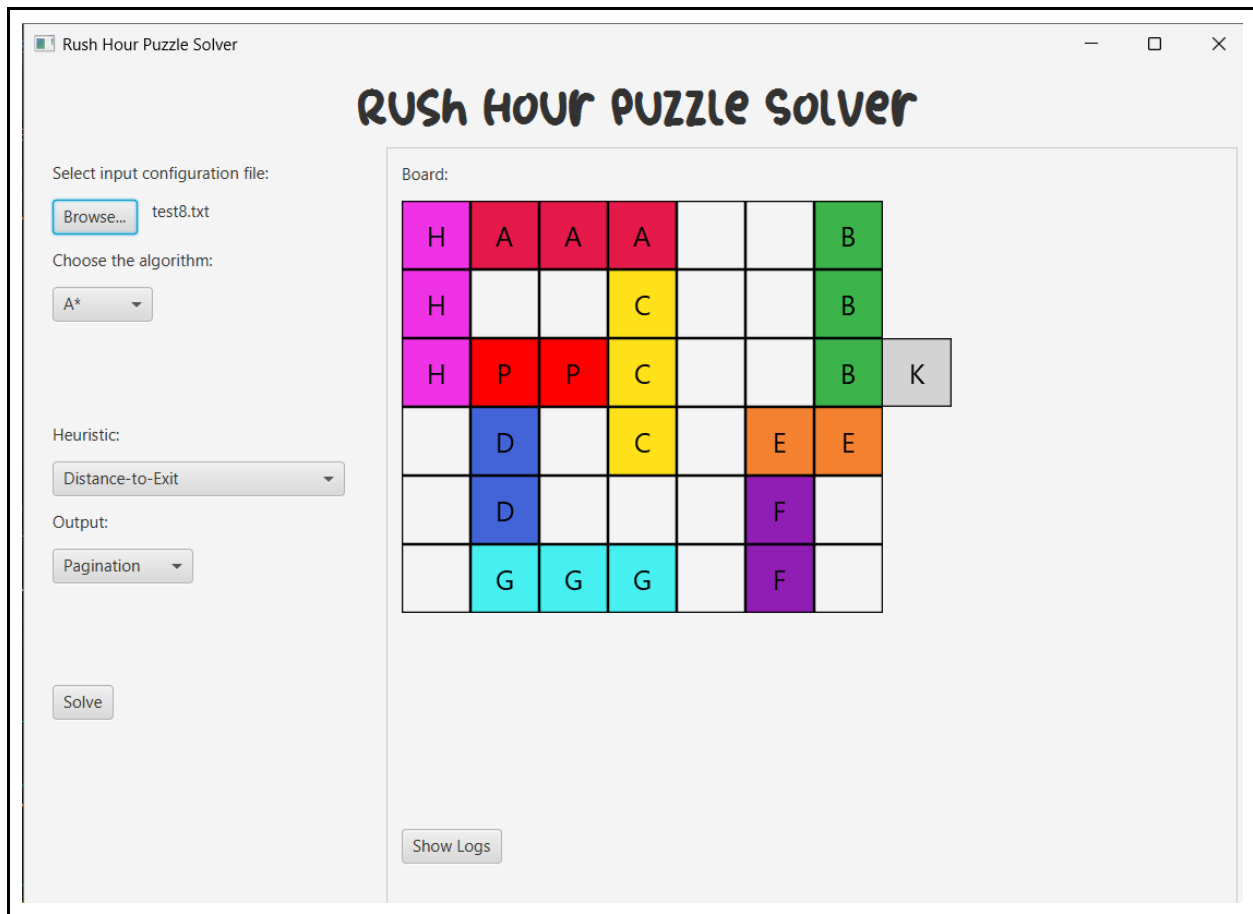
## 5.22. Test Case 22

Isi test8.txt

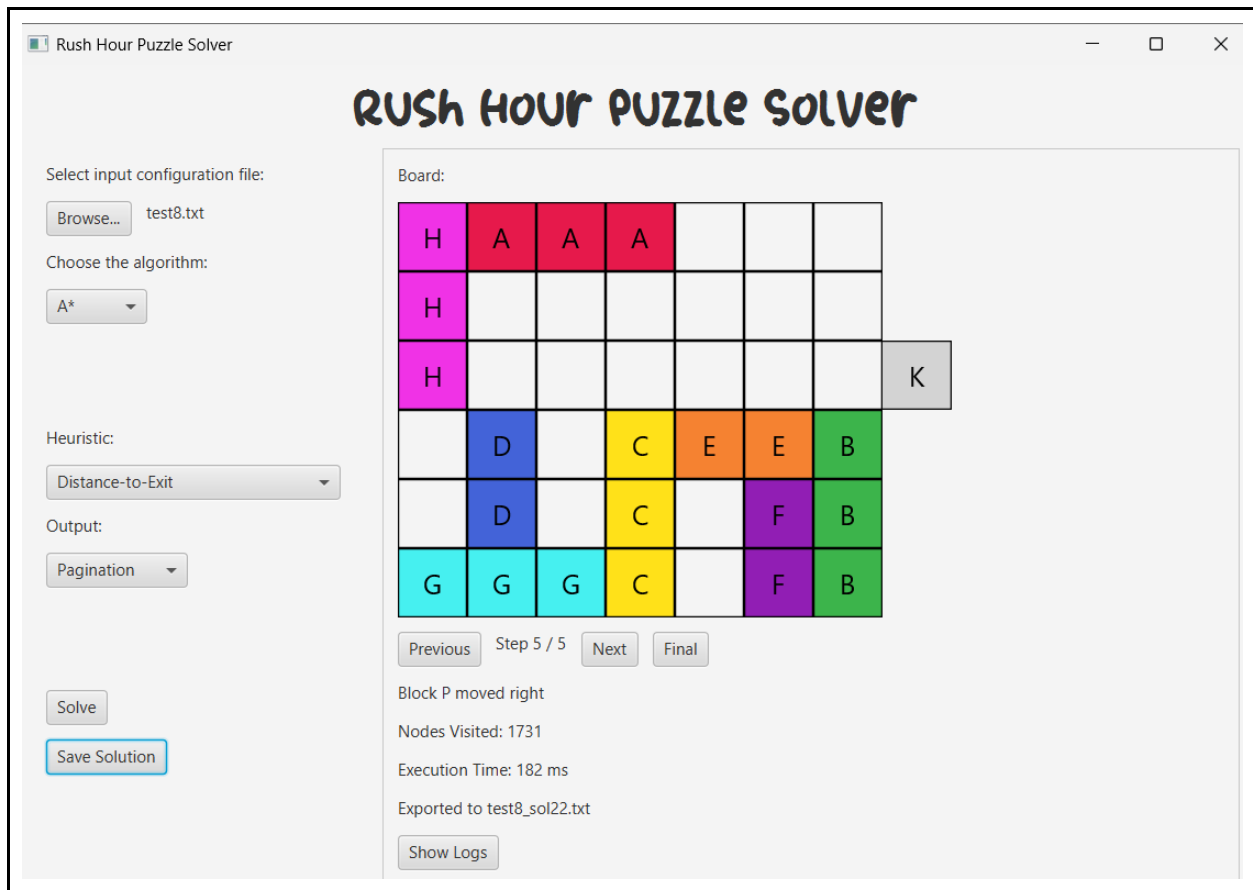


```
test8.txt U X
test > test8.txt
1 6 7
2 8
3 HAAA..B
4 H..C..B
5 HPPC..BK
6 .D.C.EE
7 .D...F.
8 .GGG.F.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test8.txt pada GUI*



*Output penyelesaian program test8.txt yang telah disimpan pada file test8\_sol22.txt*

Initial state

Step 0:

HAAA..B

H..C..B

HPPC..BK

.D.C.EE

.D...F.

.GGG.F.

.  
.  
.

Step 5:

Block P moved right

HAAA...

H.....

H.....K

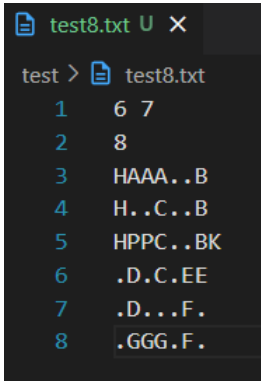
.D.CEEB

.D.C.FB  
GGGC.FB

Nodes Visited: 1731  
Execution Time: 182 ms

### 5.23. Test Case 23

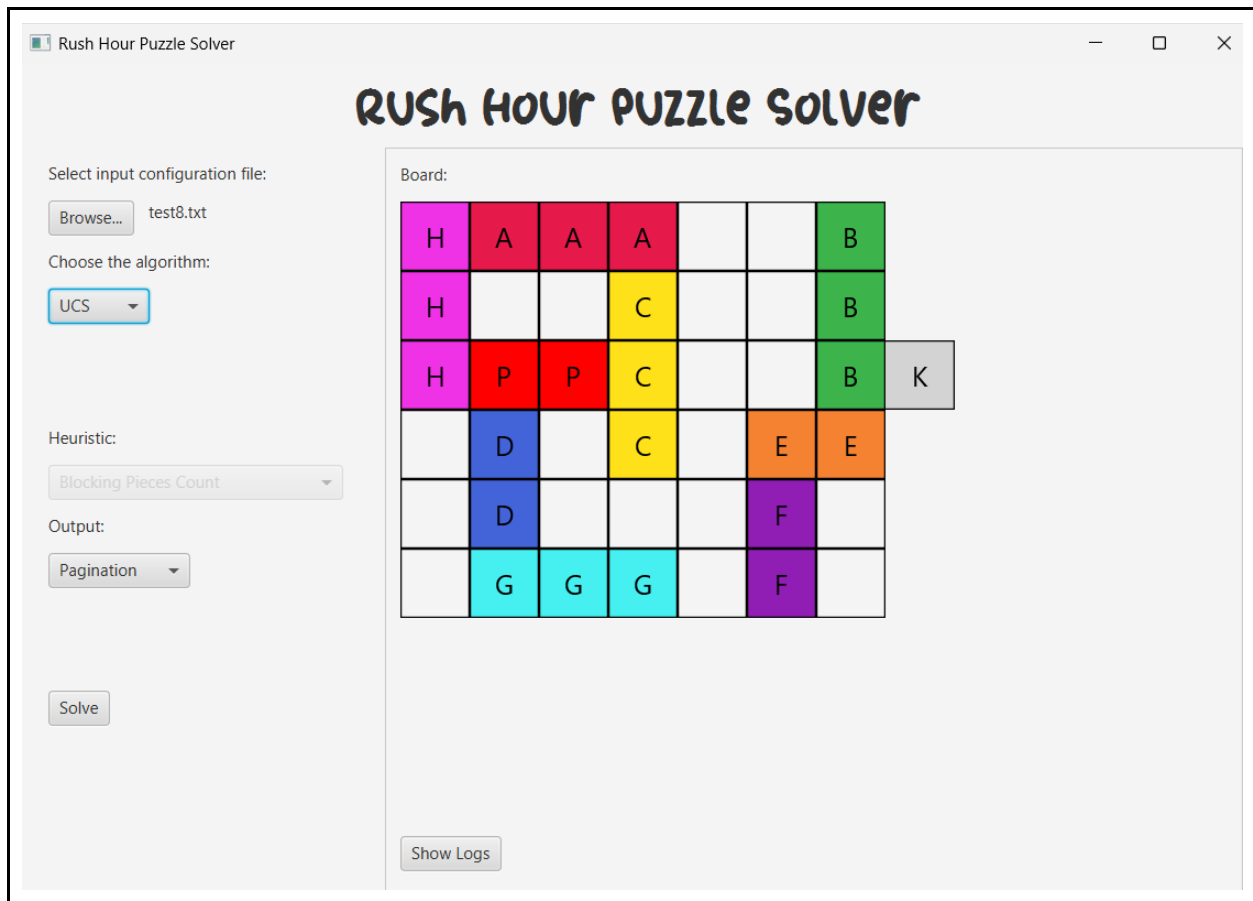
Isi test8.txt



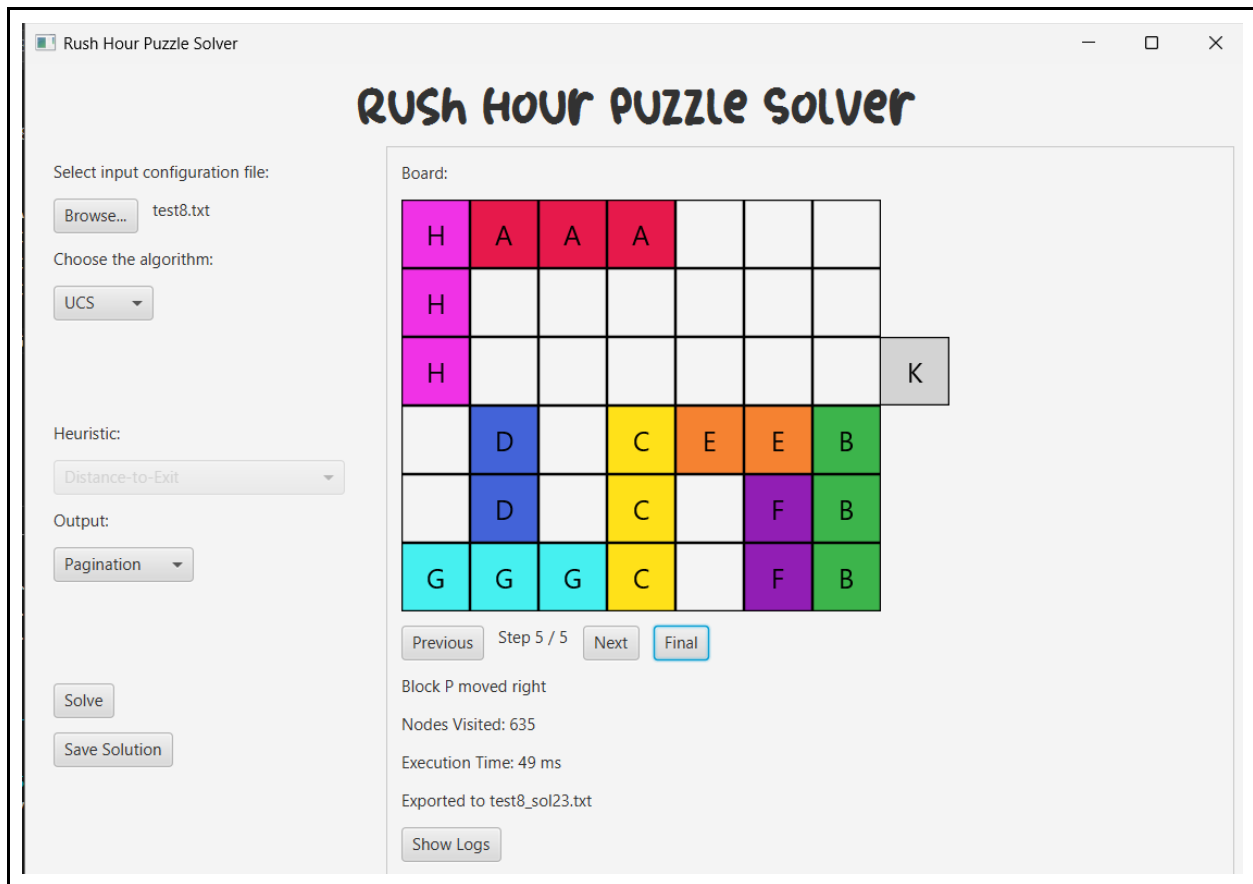
```
test8.txt U X
test > test8.txt
1 6 7
2 8
3 HAAA..B
4 H..C..B
5 HPPC..BK
6 .D.C.EE
7 .D...F.
8 .GGG.F.
```

Tampilan pertama saat program baru dijalankan





*Output penyelesaian program test8.txt pada GUI*



*Output* penyelesaian program test8.txt yang telah disimpan pada file test8\_sol23.txt

Initial state

Step 0:

HAAA..B

H..C..B

HPPC..BK

.D.C.EE

.D...F.

.GGG.F.

.

.

.

Step 5:

Block P moved right

HAAA...

H.....

H.....K

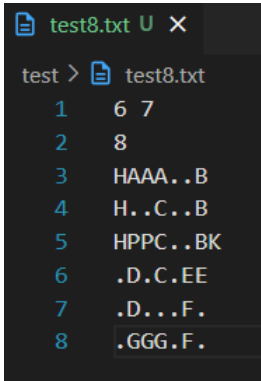
.D.CEEB

.D.C.FB  
GGGC.FB

Nodes Visited: 635  
Execution Time: 49 ms

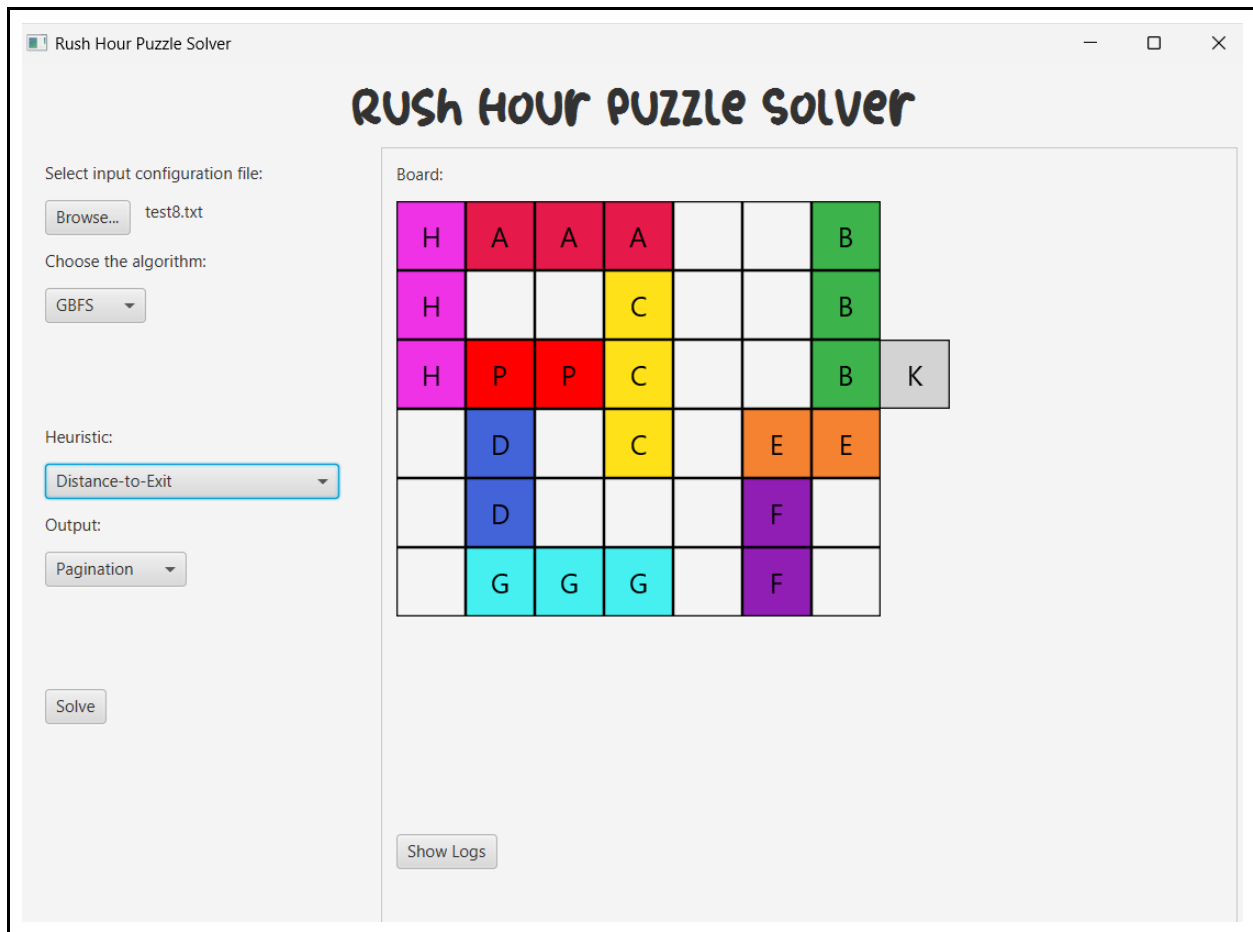
#### 5.24. Test Case 24

Isi test8.txt

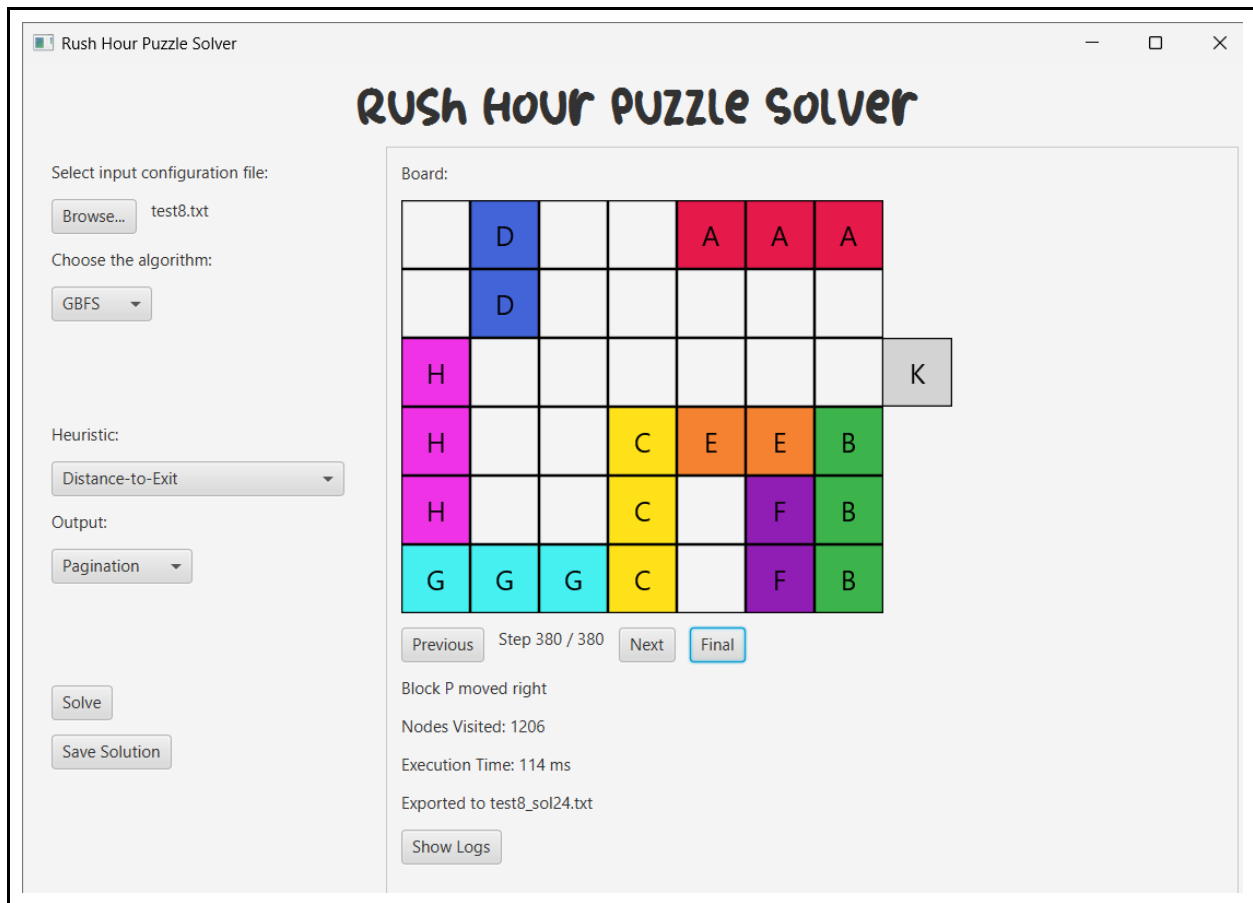


```
test8.txt U X
test > test8.txt
1 6 7
2 8
3 HAAA..B
4 H..C..B
5 HPPC..BK
6 .D.C.EE
7 .D...F.
8 .GGG.F.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test8.txt pada GUI*



*Output* penyelesaian program test8.txt yang telah disimpan pada file test8\_sol24.txt

Initial state

Step 0:

HAAA..B

H..C..B

HPPC..BK

.D.C.EE

.D...F.

.GGG.F.

.

.

.

Step 380:

Block P moved right

.D..AAA

.D.....

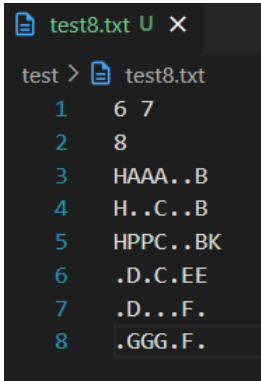
H.....K

H..CEEB  
H..C.FB  
GGGC.FB

Nodes Visited: 1206  
Execution Time: 114 ms

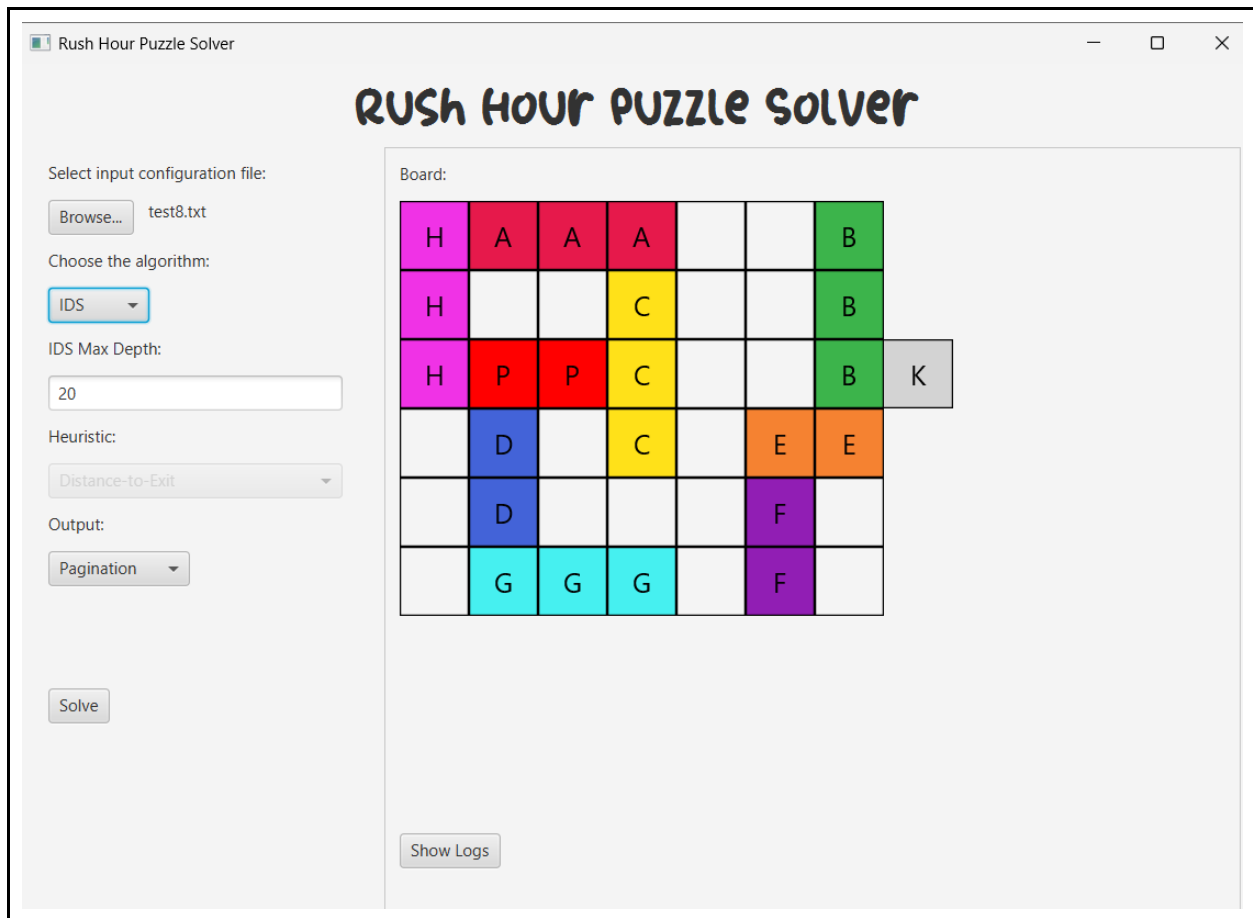
### 5.25. Test Case 25

Isi test8.txt

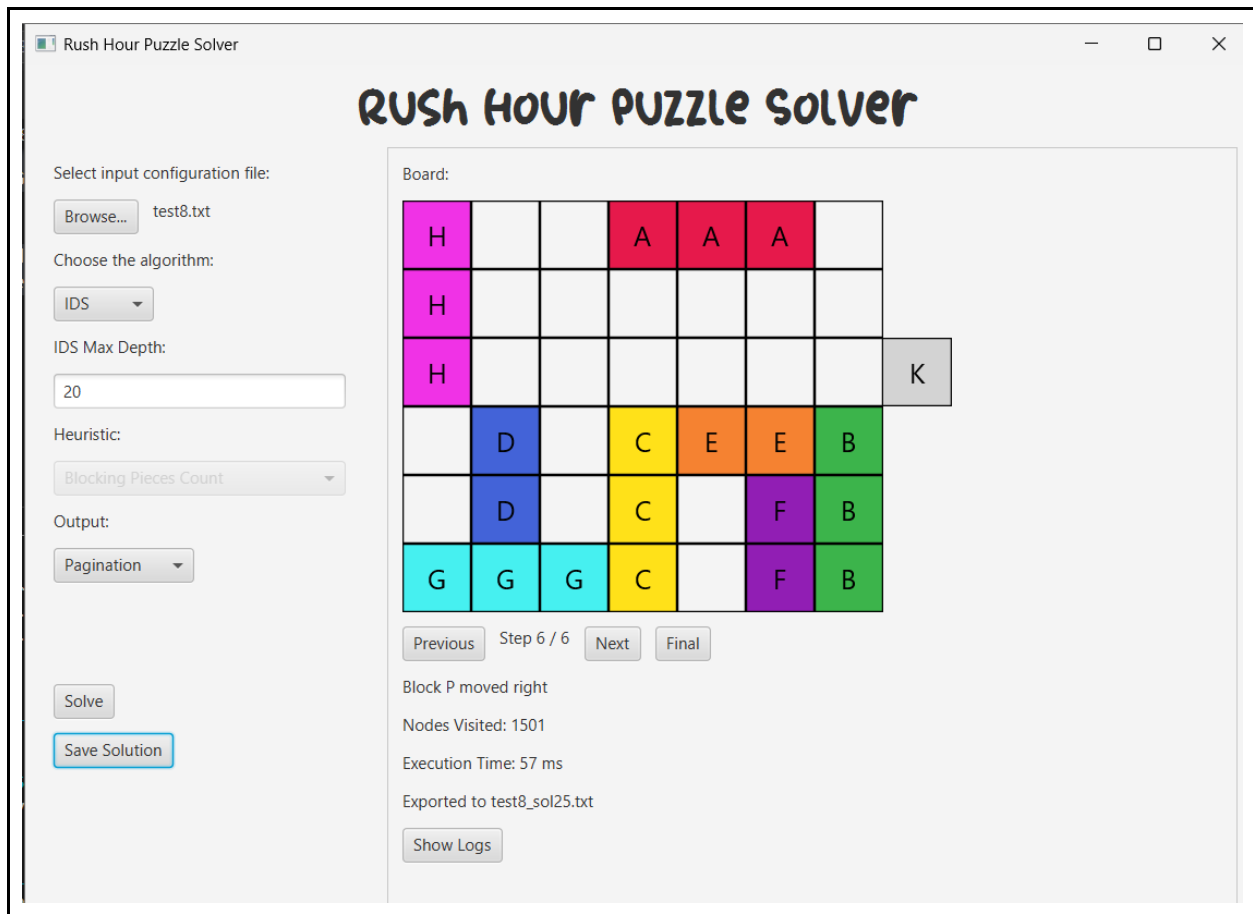


```
test8.txt U X
test > test8.txt
1 6 7
2 8
3 HAAA..B
4 H..C..B
5 HPPC..BK
6 .D.C.EE
7 .D...F.
8 .GGG.F.
```

Tampilan pertama saat program baru dijalankan



*Output penyelesaian program test8.txt pada GUI*



*Output penyelesaian program test8.txt yang telah disimpan pada file test8\_sol25.txt*

Initial state

Step 0:

HAAA..B

H..C..B

HPPC..BK

.D.C.EE

.D...F.

.GGG.F.

.

.

.

Step 6:

Block P moved right

H..AAA.

H.....

H.....K



.D.CEEB  
.D.C.FB  
GGGC.FB

Nodes Visited: 1501  
Execution Time: 57 ms

## BAB VI

### ANALISIS ALGORITMA

#### 6.1. Algoritma *Pathfinding Greedy Best First Search*

GBFS adalah algoritma pencarian jalur (*pathfinding*) yang mengandalkan fungsi heuristik  $h(n)$  sebagai estimasi jarak dari simpul saat ini ke tujuan, dan memilih node dengan nilai  $h(n)$  terkecil. Berbeda dari algoritma seperti  $A^*$ , GBFS mengabaikan biaya aktual dari awal ke simpul saat ini ( $g(n)$ ), sehingga pada GBFS dapat dikatakan bahwa  $g(n) = 0$  atau tidak dipertimbangkan sama sekali. Algoritma ini hanya mengejar tujuan secepat mungkin berdasarkan estimasi heuristik.

Metode `solve()` bekerja dengan menggunakan *PriorityQueue* untuk menyimpan dan mengambil Board dengan nilai heuristik terendah terlebih dahulu. Setiap iterasi, simpul (*board*) dengan nilai heuristik terendah diambil dari antrian. Jika simpul tersebut merupakan solusi (yaitu *goal state*), maka jalur solusi dibentuk kembali melalui metode `reconstructPath()`, yang melacak parent dari simpul hingga awal. Jika belum mencapai tujuan, algoritma menghasilkan semua tetangganya (*neighbors*), menyetelnya sebagai anak dari simpul saat ini, dan menambahkannya ke antrian prioritas, tanpa memedulikan apakah simpul tersebut pernah dikunjungi dan melakukan pengecekan *visited* dilakukan setelah *polling* dari *queue*. Selama proses, jumlah simpul yang dikunjungi dan waktu eksekusi dicatat sebagai metrik kinerja.

Secara keseluruhan, algoritma *Greedy Best First Search* (GBFS) cepat dan efisien dalam kasus-kasus tertentu, terutama jika heuristik yang digunakan cukup baik dalam memperkirakan jarak ke solusi. Namun, karena GBFS hanya mempertimbangkan fungsi heuristik  $h(n)$  dan mengabaikan  $g(n)$  (biaya dari titik awal ke node saat ini), algoritma ini rentan terhadap beberapa masalah mendasar. Pertama, GBFS tidak menjamin kelengkapan (*not complete*), artinya bisa gagal menemukan solusi meskipun sebenarnya solusi ada, terutama jika struktur pencariannya menyebabkan ia tidak menjelajahi semua kemungkinan jalur. Kedua, algoritma ini bisa terjebak di local minima atau dataran (*plateau*), yaitu ketika heuristik tidak memberikan panduan arah yang jelas, sehingga eksplorasi menjadi stagnan. Ketiga, keputusan GBFS bersifat tidak dapat dibatalkan (*irrevocable*); ketika algoritma memilih satu jalur karena heuristiknya terlihat menjanjikan,

ia tidak akan kembali untuk mengevaluasi jalur lain yang mungkin sebenarnya lebih optimal. Oleh karena itu, meskipun heuristik dapat mempercepat pencarian, penting untuk menggabungkan heuristik ke dalam pencarian sistematis seperti yang dilakukan pada algoritma A\* agar pencarian tetap efisien namun juga optimal dan lengkap.

## 6.2. Algoritma *Pathfinding UCS (Uniform Cost Search)*

UCS hanya mempertimbangkan biaya aktual dari awal ke simpul saat ini ( $g(n)$ ) sebagai fungsi evaluasinya, yaitu  $f(n) = g(n)$ , tanpa menggunakan estimasi heuristik. Dalam konteks implementasi ini, setiap perpindahan dari satu konfigurasi papan ke tetangganya dianggap memiliki biaya tetap sebesar 1, sehingga  $g(n)$  merepresentasikan jumlah langkah atau depth dari simpul awal ke simpul saat ini. Proses pencarian dimulai dari simpul awal yang dimasukkan ke dalam PriorityQueue, ketika elemen diprioritaskan berdasarkan biaya kumulatif terkecil. Sebuah *inner class* bernama *Node* digunakan untuk menyimpan objek *Board* beserta nilai *cost*-nya. Selama proses pencarian, simpul dengan biaya terendah akan diambil dari antrian. Jika simpul tersebut merupakan *goal state*, maka algoritma membangun ulang jalur solusi dengan menelusuri pointer *parent* dari simpul hingga awal, dan mengembalikan jalurnya sebagai hasil. Jika simpul tersebut belum mencapai tujuan dan belum pernah dikunjungi, maka tetangga-tetangganya dihasilkan, diberi parent yang sesuai, dan dimasukkan ke dalam antrian dengan biaya yang ditambahkan satu dari simpul induknya. Algoritma akan terus berjalan hingga solusi ditemukan atau semua kemungkinan telah dieksplorasi. Hal tersebut dapat terjadi karena UCS mempertimbangkan  $g(n)$  secara penuh dan tidak menggunakan heuristik, algoritma ini menjamin solusi optimal dalam hal jumlah langkah, meskipun bisa menjadi lambat jika ruang pencariannya sangat besar.

Pada penyelesaian *puzzle Rush Hour*, algoritma UCS (*Uniform Cost Search*) akan menghasilkan urutan node yang dibangkitkan dan path yang dihasilkan sama seperti BFS (*Breadth-First Search*), dengan satu syarat penting: setiap langkah (perpindahan dari satu *board* ke *board* tetangga) memiliki biaya yang sama, yaitu konstan. Dalam kode UCSSolver di atas, setiap *neighbor* selalu diberi tambahan biaya sebesar 1 ( $\text{current.cost} + 1$ ), yang artinya semua langkah dianggap memiliki *cost* yang sama. Dalam kondisi seperti ini, UCS secara efektif berperilaku sama seperti BFS, karena UCS akan mengeksplorasi simpul-simpul berdasarkan urutan *cost* terkecil terlebih dahulu yang artinya berdasarkan

kedalaman level (jumlah langkah dari *start*), seperti halnya BFS. Oleh karena itu, jika setiap langkah dalam Rush Hour diberi bobot yang sama, maka UCS dan BFS akan mengunjungi *node-node* dalam urutan yang sama, dan juga menghasilkan solusi (*path* menuju *goal*) yang identik. Perbedaan baru akan muncul jika biaya tiap langkah berbeda dalam kasus tersebut UCS tetap optimal, sementara BFS tidak lagi menjamin solusi dengan *cost* terkecil.

### 6.3. Algoritma *Pathfinding A\** (*A-Star*)

Algoritma *A\** digunakan untuk menyelesaikan puzzle Rush Hour dengan menggabungkan biaya riil dari awal ke simpul saat ini dan estimasi jarak tersisa ke tujuan. Fungsi evaluasi yang dipakai adalah  $f(n)=g(n)+h(n)$ , dengan  $g(n)$  adalah jumlah langkah yang telah ditempuh (bertambah 1 setiap kali berpindah state) dan  $h(n)$  dihitung oleh *Heuristic.evaluate* sesuai nama heuristik yang diberikan ketika objek dibuat. Metode *solve()* memelihara open list berupa *PriorityQueue<Node>* yang selalu mem-poll simpul dengan nilai  $f$  terkecil terlebih dahulu. Setiap simpul di-poll dihitung sebagai simpul yang dikunjungi, kemudian dicek apakah sudah mencapai keadaan *goal*; bila ya, jalur solusi dibangun kembali lewat *reconstructPath()* dengan menelusuri atribut *parent*. Jika belum, simpul dimasukkan ke *visited* set agar tidak diproses ulang, lalu seluruh *neighbor*-nya dihasilkan; tiap *neighbor* diberi pointer *parent* ke simpul asal, biaya  $g$ -nya diperbaharui menjadi  $\text{curr.g} + 1$ , estimasi  $h$  dihitung ulang, dan objek *Node* baru berisi  $(g,f)$  dimasukkan kembali ke antrian. Proses ini berulang hingga *state goal* ditemukan atau *open list* kosong (yang berarti tidak ada solusi). Dengan memadukan  $g$  dan  $h$  yang konsisten, algoritma *A\** menjamin menemukan jalur dengan biaya minimum sambil tetap efisien karena arahnya dipandu oleh heuristik.

Heuristik yang digunakan pada algoritma *A\** adalah *admissible* jika dan hanya jika tidak pernah melebihi biaya sebenarnya (biaya minimum) dari simpul saat ini menuju simpul tujuan. Berdasarkan definisi *admissible heuristic* dalam salindia kuliah, heuristik harus optimistik yaitu selalu memperkirakan biaya secara kurang atau sama dengan biaya sebenarnya. Dalam implementasi algoritma *A\** di atas, nilai heuristik dihitung menggunakan *Heuristic.evaluate(neighbor, heuristicName)*, dan nilai total yang digunakan dalam antrian prioritas (*PriorityQueue*) adalah  $f = g + h$ , dengan  $g$  adalah biaya dari awal

hingga simpul saat ini, dan  $h$  adalah estimasi biaya dari simpul saat ini ke tujuan. Jika fungsi *Heuristic.evaluate* yang dipakai menjamin bahwa nilai  $h$  selalu kurang dari atau sama dengan jarak sebenarnya ke goal, maka heuristik tersebut dapat dikatakan *admissible*. Oleh karena itu, apakah heuristik pada A\* dalam kode ini *admissible* sangat bergantung pada implementasi konkret dari metode *Heuristic.evaluate*. Jika metode tersebut sesuai dengan definisi *admissible* dari salindia kuliah, maka algoritma A\* yang diimplementasikan ini pun menggunakan heuristik yang *admissible*.

Secara teoritis, algoritma A\* lebih efisien dibandingkan dengan algoritma UCS dalam penyelesaian puzzle Rush Hour, asalkan heuristik yang digunakan bersifat *admissible* dan konsisten. UCS pada dasarnya adalah versi A\* dengan heuristik nol ( $h(n) = 0$ ), sehingga UCS mengeksplorasi semua kemungkinan berdasarkan jarak dari start ( $g(n)$ ) tanpa mempertimbangkan seberapa dekat sebuah state ke goal. Sebaliknya, A\* menggabungkan informasi dari  $g(n)$  dan  $h(n)$  untuk memprioritaskan eksplorasi pada state-state yang diperkirakan lebih dekat ke solusi. Dengan heuristik yang baik, A\* dapat menghindari banyak eksplorasi tidak perlu yang dilakukan oleh UCS, sehingga mengunjungi lebih sedikit simpul secara keseluruhan. Oleh karena itu, secara teoritis A\* lebih efisien karena mampu menyeimbangkan antara eksplorasi mendalam dan kedekatan ke tujuan, yang membuatnya lebih cepat menemukan solusi optimal dalam banyak kasus, termasuk pada Rush Hour. Namun, keunggulan ini sangat tergantung pada kualitas heuristik yang digunakan.

#### 6.4. Algoritma *Pathfinding IDS (Iterative Deepening Search)*

Algoritma pencarian *Iterative Deepening Search* (IDS) digunakan untuk menyelesaikan puzzle Rush Hour. IDS menggabungkan kelebihan dari pencarian *Depth-First Search* (DFS) yang hemat memori dan *Breadth-First Search* (BFS) yang menjamin solusi optimal pada graf tak berbobot, dengan cara melakukan DFS secara berulang namun membatasi kedalaman pencarian di setiap iterasinya. Metode *solve()* menginisialisasi pencarian dengan kedalaman terbatas mulai dari 0 hingga batas maksimum (*maxDepth*) yang ditentukan saat instansiasi objek. Untuk setiap nilai kedalaman tersebut, ia memanggil fungsi *dfs()*, yang melakukan pencarian mendalam hingga kedalaman tersebut. Pada setiap pemanggilan, simpul yang sedang dieksplorasi dicek apakah merupakan tujuan (*goal*

*state*); jika ya, maka jalur solusi dibentuk kembali melalui parent-pointer menggunakan *reconstructPath()*. Jika belum, fungsi akan merekursifkan DFS pada semua tetangga simpul saat ini yang belum dikunjungi, dengan mengurangi kedalaman yang tersisa. Jika tidak ada solusi ditemukan pada suatu iterasi, kedalaman pencarian akan ditingkatkan dan proses diulang. Meskipun IDS berpotensi mengunjungi simpul yang sama berkali-kali karena *restart* DFS setiap iterasi, keunggulannya terletak pada efisiensi memori dan kemampuan menemukan solusi optimal ketika biaya langkah seragam.

## 6.5. Algoritma Heuristik

Heuristik dalam konteks algoritma pencarian seperti A\* adalah fungsi estimasi yang digunakan untuk memperkirakan biaya minimum dari suatu state ke goal. Dalam penyelesaian puzzle seperti *Rush Hour*, heuristik dirancang untuk membantu algoritma menentukan prioritas eksplorasi state, agar lebih efisien menemukan solusi. Heuristik yang *admissible* tidak pernah melebihi biaya sebenarnya ke goal, dan yang konsisten menjamin bahwa biaya antar node tidak melanggar aturan segitiga. Tiga heuristik yang digunakan dalam kode ini adalah: *Blocking Pieces Count*, *Blocking Pieces With Movability*, dan *Distance to Exit*. Masing-masing memiliki kelebihan dan pendekatan berbeda dalam memperkirakan jarak ke solusi.

Heuristik *Blocking Pieces Count* menghitung jumlah kendaraan yang menghalangi jalur langsung kendaraan utama (berlabel 'P') menuju pintu keluar 'K'. Dengan asumsi bahwa kendaraan 'P' harus bergerak lurus ke arah keluar (baik secara horizontal maupun vertikal), heuristik ini menghitung berapa banyak kendaraan lain yang berada di jalur tersebut. Semakin banyak kendaraan yang menghalangi, semakin besar estimasi biaya ke solusi. Heuristik ini cukup sederhana dan bersifat *admissible*, karena hanya menghitung hambatan langsung tanpa melebih-lebihkan biaya yang sebenarnya.

Heuristik *Blocking Pieces With Movability* merupakan pengembangan dari heuristik sebelumnya, namun dengan menambahkan penalti jika kendaraan penghalang tidak dapat digeser untuk memberi jalan. Selain menghitung jumlah kendaraan yang menghalangi jalur 'P' menuju 'K', heuristik ini juga memeriksa apakah masing-masing kendaraan penghalang tersebut bisa bergerak ke atas/bawah (jika kendaraan 'P' horizontal) atau ke

kiri/kanan (jika 'P' vertikal). Jika kendaraan penghalang tidak bisa digerakkan, maka heuristik memberikan penalti tambahan, menandakan bahwa state tersebut kemungkinan lebih sulit diselesaikan. Heuristik ini tetap *admissible* jika penalti tidak melebihi estimasi biaya minimum yang sebenarnya.

Heuristik *Distance-to-Exit* menghitung jarak garis lurus dari bagian depan kendaraan 'P' ke pintu keluar 'K', berdasarkan arah gerak kendaraan (horizontal atau vertikal). Jika kendaraan 'P' tidak berada di jalur yang sejajar dengan pintu keluar, maka fungsi ini mengembalikan nilai maksimum sebagai indikasi bahwa state tersebut tidak mengarah langsung ke solusi. Heuristik ini berfokus pada jarak spasial daripada hambatan, dan meskipun sangat ringan secara komputasi, ia bisa menjadi kurang akurat dibanding dua heuristik sebelumnya karena tidak mempertimbangkan hambatan kendaraan lain. Namun, heuristik ini juga *admissible* karena tidak pernah melebihi biaya nyata untuk mencapai solusi.

## BAB VII

### KESIMPULAN DAN SARAN

#### 7.1. Kesimpulan

*Puzzle Rush Hour Solver* ini telah berhasil dikembangkan menggunakan bahasa Java dan antarmuka grafis berbasis JavaFX, dengan fokus pada pencarian solusi melalui algoritma pathfinding. Pengguna diberikan pilihan untuk menyelesaikan puzzle menggunakan tiga algoritma utama, yaitu *Uniform Cost Search* (UCS), *Greedy Best First Search* (GBFS), dan A\* (A-Star), serta satu algoritma tambahan yakni *Iterative Deepening Search* (IDS), yang memerlukan input kedalaman eksplorasi dari pengguna. Selain itu, pengguna dapat memilih jenis heuristik yang digunakan dalam algoritma GBFS dan A\*, seperti *Blocking Pieces Count*, *Blocking Pieces With Movability*, dan *Distance-to-Exit* sebagai fitur bonus. Aplikasi ini juga menyediakan dua mode tampilan solusi, yaitu secara paginasi maupun animasi, yang memberikan fleksibilitas dan pengalaman visual yang interaktif dalam memahami proses pencarian jalur solusi pada *puzzle Rush Hour*.

Berdasarkan *testing* yang telah dilakukan menggunakan test1.txt, algoritma yang paling cepat adalah menggunakan algoritma *Greedy Best First Search* (GBFS) karena jumlah simpul yang dikunjungi cenderung lebih sedikit sehingga waktu eksekusi yang dibutuhkan juga semakin singkat. Berdasarkan *testing* yang telah dilakukan menggunakan test2.txt, heuristik yang paling sangkil adalah *Blocking Pieces With Movability* karena jumlah simpul yang dikunjungi cenderung lebih sedikit sehingga waktu eksekusi yang dibutuhkan juga semakin singkat.

#### 7.2. Saran

Untuk pengembangan lebih lanjut, *puzzle Rush Hour Solver* ini dapat ditingkatkan dari segi performa dan pengalaman pengguna. Salah satu saran perbaikan adalah dengan mengoptimalkan efisiensi pencarian, terutama untuk algoritma IDS yang cenderung lambat pada kedalaman besar. Selain itu, penambahan visualisasi jalur solusi secara interaktif, seperti penanda langkah atau animasi gerakan kendaraan, akan memberikan pemahaman yang lebih baik bagi pengguna. Fitur penyimpanan dan pemuatan konfigurasi papan secara dinamis juga dapat ditambahkan untuk meningkatkan fleksibilitas penggunaan. Terakhir, pengujian aplikasi pada berbagai ukuran papan dan variasi kompleksitas kendaraan sangat disarankan agar sistem menjadi lebih *robust* dan siap digunakan secara luas.



### 7.3. Komentor

Kami mengucapkan terima kasih kepada asisten-asisten yang bertanggung jawab atas pelaksanaan tugas besar ini dan dosen-dosen mata kuliah IF2211 Strategi Algoritma yang telah memberikan wawasan dan pengetahuan kepada kami saat perkuliahan. Kami juga mengucapkan terima kasih kepada mahasiswa Teknik Informatika ITB lain yang telah memberikan semangat dan dukungan sehingga pada akhirnya tugas besar ini dapat selesai dengan baik.

### 7.4. Refleksi

Tugas besar ini telah memberikan pemahaman yang lebih mendalam mengenai penerapan algoritma pencarian dalam pemecahan masalah nyata, khususnya dalam konteks game berbasis logika. Implementasi algoritma seperti UCS, GBFS, A\*, dan IDS yang menuntut pemahaman tidak hanya secara teoritis, tetapi juga dari sisi efisiensi dan struktur data yang sesuai. Selain itu, integrasi dengan GUI menggunakan JavaFX memberikan tantangan tersendiri dalam hal sinkronisasi visual dengan logika program. Melalui proyek ini, saya belajar pentingnya perencanaan desain program, debugging yang sistematis, serta pentingnya pengalaman pengguna (*user experience*) dalam membangun aplikasi yang interaktif dan informatif. Proyek ini menjadi pengalaman berharga dalam menggabungkan teori algoritma dengan pengembangan perangkat lunak yang nyata.

## DAFTAR PUSTAKA

- Munir, R. 2025. “Tugas Kecil 3: Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding”. [Online]. Tersedia: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/Tucil3-Stima-2025.pdf>. [17 Mei 2025].
- Munir, R. 2025. “Route Planning (2025) Bagian 1”. [Online]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf). [20 Mei 2025].
- Munir, R. 2025. “Route Planning (2025) Bagian 2”. [Online]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf). [20 Mei 2025].

## LAMPIRAN

### Lampiran 1. Tautan *Repository*

[https://github.com/naomirisaka/Tucil3\\_13523019\\_13523122](https://github.com/naomirisaka/Tucil3_13523019_13523122)

### Lampiran 2. Tabel Checklist Program

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