IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding
Laporan Tugas Kecil 3

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma pada Semester 4 (empat) Tahun Akademik 2024/2025

Disusun oleh:

Muhammad Farrel Wibowo (13523153)

Mahesa Fadhillah Andre (13523140)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

Daftar Isi

BAB I

DESKRIPSI TUGAS

Gambar 1. Rush Hour Puzzle
(Sumber: https://www.thinkfun.com/en-US/products/educational-games/rush-hour-76582)

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

Komponen penting dari permainan Rush Hour terdiri dari:

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

2. Piece - *Piece* adalah sebuah kendaraan di dalam papan. Setiap *piece* memiliki *posisi*, *ukuran*, dan *orientasi*. *Orientasi* sebuah *piece* hanya dapat berupa horizontal atau vertikal–tidak mungkin diagonal. *Piece* dapat memiliki beragam *ukuran*, yaitu jumlah *cell* yang ditempati oleh *piece*. Secara standar, variasi *ukuran* sebuah *piece* adalah *2-piece* (menempati 2 *cell*) atau *3-piece* (menempati 3 *cell*). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

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

4. Pintu Keluar - *Pintu keluar* adalah tempat *primary piece* dapat digerakkan keluar untuk menyelesaikan permainan

5. Gerakan - *Gerakan* yang dimaksudkan adalah pergeseran *piece* di dalam permainan. *Piece* hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

Spesifikasi Tugas Kecil 3:

- Buatlah program sederhana dalam bahasa C/C++/Java/Javascript yang mengimplementasikan algoritma *pathfinding Greedy Best First Search, UCS (Uniform Cost Search), dan A** dalam menyelesaikan permainan Rush Hour.
- Tugas dapat dikerjakan individu atau berkelompok dengan anggota maksimal 2 orang (sangat disarankan). Boleh lintas kelas dan lintas kampus, tetapi tidak boleh sama dengan anggota kelompok pada tugas kecil Strategi Algoritma sebelumnya.
- Algoritma *pathfinding* minimal menggunakan satu *heuristic* (2 atau lebih jika mengerjakan *bonus*) yang ditentukan sendiri. Jika mengerjakan *bonus*, *heuristic* yang digunakan ditentukan berdasarkan input pengguna.
- Algoritma dijalankan secara terpisah. Algoritma yang digunakan ditentukan berdasarkan Input pengguna.
- Alur Program:
    1. [INPUT] konfigurasi permainan/test case dalam format ekstensi .txt. File *test case* tersebut berisi:
        1. Dimensi Papan terdiri atas dua buah variabel A dan B yang membentuk papan berdimensi AxB
        2. Banyak *piece* BUKAN *primary piece* direpresentasikan oleh variabel integer N.
        3. Konfigurasi papan yang mencakup penempatan *piece* dan *primary piece*, serta lokasi *pintu keluar*. *Primary Piece* dilambangkan dengan huruf P dan pintu keluar dilambangkan dengan huruf K. *Piece* dilambangkan dengan huruf dan karakter selain *P* dan *K*, dan huruf/karakter berbeda melambangkan *piece*

yang berbeda. *Cell* kosong dilambangkan dengan karakter '.' (titik). (Catatan: ingat bahwa *pintu keluar* pasti berada di *dinding* papan dan sejajar dengan orientasi *primary piece*)

File .txt yang akan dibaca memiliki format sebagai berikut:

```
A B
N
konfigurasi_papan
```

Contoh Input

```
6 6
11
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.
```

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

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

```
  K          X        XXX
 XXX         X        XXX
 XXX         X        XXX
 XXX         X          K
            KXX
             X

             X
             X
             X
```

2. [INPUT] algoritma *pathfinding* yang digunakan
3. [INPUT] *heuristic* yang digunakan (bonus)
4. [OUTPUT] Banyaknya gerakan yang diperiksa (alias banyak 'node' yang dikunjungi)
5. [OUTPUT] Waktu eksekusi program

6. [OUTPUT] konfigurasi *papan* pada setiap tahap pergerakan/pergeseran. Output ini tidak harus diimplementasi apabila mengerjakan *bonus output GUI*. Gunakan print berwarna untuk menunjukkan pergerakan *piece* dengan jelas. Cukup mewarnakan *primary piece, pintu keluar,* dan *piece* yang digerakkan saja (boleh dengan *highlight* atau *text color*). Pastikan ketiga komponen tersebut memiliki warna berbeda.

   Format sekuens adalah sebagai berikut:

```
Papan Awal
[konfigurasi_papan_awal]

Gerakan 1: [piece]-[arah gerak]
[konfigurasi_papan_gerakan_1]

Gerakan 2: [piece]-[arah gerak]
[konfigurasi_papan_gerakan_2]

Gerakan [N]: [piece]-[arah gerak]
[konfigurasi_papan_gerakan_N]
dst
```

   Contoh Output

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

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

Gerakan 2: F-bawah
AAB...
..BCDF
```

```
GPPCDFK
GHIIIF
GHJ...
LLJMM.
dst
```

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

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

BAB II
LANDASAN TEORI

2.1. UCS

Uniform Cost Search (UCS) adalah algoritma pencarian yang termasuk dalam kategori blind search atau pencarian buta, yang berarti algoritma ini tidak menggunakan informasi heuristik tentang jarak ke tujuan. Tujuan utama UCS adalah mencari lintasan dengan total biaya paling rendah dari simpul awal menuju simpul tujuan. Berbeda dengan algoritma seperti Breadth-First Search (BFS) yang hanya mempertimbangkan jumlah langkah (depth), UCS mempertimbangkan bobot atau biaya dari setiap lintasan, sehingga jalur yang ditemukan adalah yang paling murah secara total.
UCS mengevaluasi setiap simpul berdasarkan fungsi evaluasi berikut:

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

dengan:

- $g(n)$ = akumulasi biaya dari simpul awal hingga simpul nnn.

Dalam implementasinya, UCS menggunakan struktur data priority queue yang selalu menempatkan simpul dengan nilai $g(n)$ terkecil di bagian depan. Dengan demikian, simpul dengan biaya total terendah akan diekspansi terlebih dahulu. UCS menjamin hasil pencarian yang lengkap (solusi pasti ditemukan jika ada) dan optimal (solusi yang ditemukan adalah dengan biaya minimum), selama semua bobot lintasan positif.

pseudocode:

```
procedure UCS(initial_state):
    priority_queue ← new PriorityQueue ordered by g(n)
    visited ← empty set
    priority_queue.add(initial_state)

    while priority_queue is not empty:
        current_state ← priority_queue.poll()
        if current_state is goal:
            return current_state
```

```
            if current_state not in visited:
                add current_state to visited
                for each neighbor in
    current_state.generateSuccessors():
                    if neighbor not in visited:
                        priority_queue.add(neighbor)

        return failure
```

1. Mulai dengan membuat *priority queue* (antrian prioritas) dan masukkan simpul awal dengan biaya nol.
2. Selama antrian belum kosong:
3. Ambil simpul dengan biaya kumulatif terkecil dari antrian (yakni simpul dengan nilai g(n) terkecil).
4. Jika simpul ini adalah simpul tujuan, maka pencarian selesai dan jalur ditemukan.
5. Tandai simpul sebagai telah dikunjungi (untuk menghindari eksplorasi ulang).
6. Ekspansi semua tetangga dari simpul ini.

a. Untuk setiap tetangga, hitung total biaya kumulatif dari awal ke simpul tersebut.
b. Jika simpul tetangga belum pernah dikunjungi atau ditemukan jalur yang lebih murah, masukkan ke antrian.

7. Jika semua simpul telah diperiksa dan simpul tujuan tidak ditemukan, maka tidak ada solusi.

2.2. Greedy Best First Search

Greedy Best First Search (GBFS) merupakan algoritma pencarian yang termasuk dalam kategori *informed search*. Berbeda dari UCS, algoritma ini hanya menggunakan fungsi heuristik untuk mengarahkan pencarian, tanpa memperhitungkan jarak yang telah ditempuh dari simpul awal. Fokus utama dari algoritma ini adalah memilih simpul yang *diperkirakan* paling dekat dengan tujuan berdasarkan nilai heuristik.

Fungsi evaluasi pada GBFS adalah:

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

dengan:

- ***h(n)*** = estimasi biaya dari simpul nnn ke tujuan.

Nilai h(n) umumnya diperoleh dari pengetahuan domain, seperti *straight-line distance* (jarak garis lurus) ke simpul tujuan dalam kasus pencarian rute. GBFS cenderung mengejar simpul dengan nilai heuristik terkecil, yang tampak paling menjanjikan secara lokal. Meskipun algoritma ini cepat dan efisien dalam menemukan jalur, ia tidak menjamin hasil yang optimal. Hal ini disebabkan karena GBFS dapat melewati jalur yang lebih panjang namun memiliki biaya keseluruhan lebih rendah, akibat hanya mengejar estimasi ke tujuan. Dalam praktiknya, GBFS dapat mengalami kegagalan akibat terjebak pada *local minimum*, *plateau*, atau bahkan jalur buntu.

Pseudocode:

```
    procedure GBFS(initial_state):
        priority_queue ← new PriorityQueue ordered by h(n)
        visited ← empty set
        priority_queue.add(initial_state)

        while priority_queue is not empty:
            current_state ← priority_queue.poll()
            if current_state is goal:
                return current_state

            if current_state not in visited:
                add current_state to visited
                for each neighbor in
    current_state.generateSuccessors():
                    if neighbor not in visited:
                        priority_queue.add(neighbor)

        return failure
```

1. Buat antrian prioritas yang mengurutkan simpul berdasarkan nilai heuristik ***h(n)***Masukkan simpul awal ke dalam antrian.
2. Selama antrian belum kosong:
3. Ambil simpul dengan nilai heuristik terkecil.
4. Jika simpul tersebut adalah tujuan, maka pencarian selesai.
5. Tandai simpul sebagai sudah dikunjungi.
6. Ekspansi semua tetangga dari simpul tersebut.

i. Untuk setiap tetangga, hitung nilai h(n) dan masukkan ke antrian jika belum pernah dikunjungi.

7. Jika tidak ada simpul tersisa dalam antrian dan belum mencapai tujuan, maka tidak ada solusi ditemukan.

2.3. A*

A* adalah algoritma pencarian yang juga termasuk dalam kategori informed search, dan merupakan gabungan dari prinsip UCS dan GBFS. Keunggulan A* adalah kemampuannya untuk mempertimbangkan baik biaya yang telah dikeluarkan dari simpul awal maupun estimasi biaya ke tujuan. Hal ini memungkinkan A* untuk mencari solusi optimal secara efisien.

Fungsi evaluasi pada A* adalah:

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

dengan:

- g(n) = biaya dari simpul awal ke simpul nnn,
- h(n) = estimasi biaya dari simpul nnn ke simpul tujuan.

A* menjamin solusi optimal jika fungsi heuristik h(n) bersifat admissible, yaitu:

$$h(n) \leq h^*(n)$$

di mana:

- $h^*(n)$ = biaya aktual minimum dari n ke tujuan.

Selain itu, heuristik juga sebaiknya consistent, yaitu memenuhi:

$$h(n) \leq c(n,n') + h(n')$$

dengan:

- c(n,n′) = biaya langsung dari simpul n ke n′.

Jika kedua sifat ini terpenuhi, A* akan selalu menemukan solusi dengan jalur total biaya minimum. A* termasuk algoritma yang lengkap dan optimal, namun memiliki kekurangan

dalam hal kebutuhan memori yang besar karena semua simpul yang pernah dibangkitkan harus disimpan.

```
procedure AStar(initial_state):
    priority_queue ← new PriorityQueue ordered by f(n) = g(n) + h(n)
    visited ← empty set
    priority_queue.add(initial_state)

    while priority_queue is not empty:
        current_state ← priority_queue.poll()
        if current_state is goal:
            return current_state

        if current_state not in visited:
            add current_state to visited
            for each neighbor in current_state.generateSuccessors():
                if neighbor not in visited:
                    priority_queue.add(neighbor)

    return failure
```

1. Buat antrian prioritas berdasarkan nilai fungsi evaluasi f(n) = g(n)+h(n), di mana:
   i. g(n): biaya dari simpul awal ke simpul saat ini.
   ii. h(n): estimasi biaya dari simpul saat ini ke tujuan.
2. Masukkan simpul awal ke antrian dengan g(n)=0.
3. Selama antrian belum kosong:
   i. Ambil simpul dengan nilai f(n) terkecil.
   ii. Jika simpul ini adalah tujuan, maka pencarian selesai dan jalur ditemukan.
   iii. Tandai simpul sebagai sudah dikunjungi.
   iv. Ekspansi semua tetangga dari simpul tersebut:
      1. Hitung nilai g(n′)=g(n)+biaya lintasan ke n′.
      2. Hitung h(n′), lalu f(n′)=g(n′)+h(n′).
      3. Jika n' belum dikunjungi atau ditemukan jalur dengan f(n′) lebih kecil, masukkan ke antrian.
4. Jika antrian habis dan tidak mencapai goal, maka tidak ditemukan solusi.

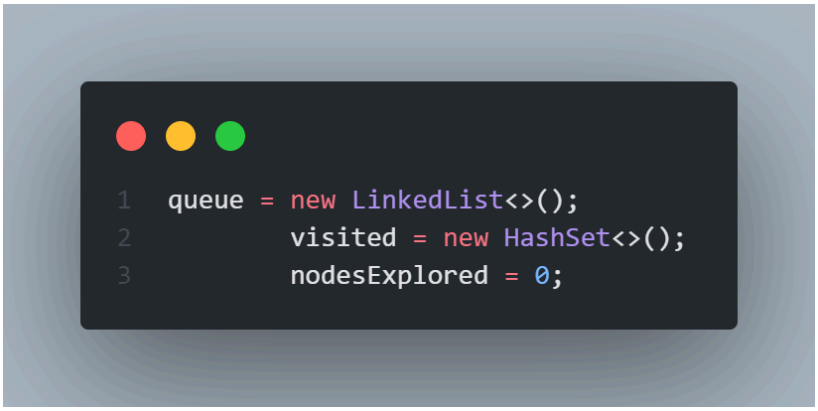
BAB III

Implementasi dan Pengujian

3.1 Implementasi dan Analisis Algoritma

3.1.1 UCS

- Frontier



- Loop utama
    1. currentState = queue.poll(): men-*dequeue* state paling lama dalam antrian
    2. Goal test: Memeriksa apakah mobil *primary* sudah mencapai sel keluar.
    3. Generate successor: Setiap mobil yang ada pada board, akan dihitung total gerakan yang dapat dilakukan oleh mobil (sesuai orientasi, jadi apabila orientasi mobil horizontal, maka akan di-cek apakah bisa geser ke kanan/ke kiri). Apabila sebuah mobil dapat bergerak, maka akan dibuat board dengan state baru setelah mobil bergerak.
    4. Jika belum pernah mengunjungi sebuah state (diperiksa oleh visited.add(moveState), maka state tersebut dimasukkan ke dalam queue lalu program menambahkan total *state* yang telah ditemukan.

- Analisis algoritma
    - *Completeness*: Selalu menemukan solusi terpendek.
    - *Optimality*: Solusi yang dihasilkan selalu minimal dalam jumlah langkahnya.

- *Time Complexity*: $O(b^d)$ dengan b adalah faktor berdasarkan rata-rata pergerakan state dan d merupakan kedalaman solusi. Sebab setiap loop menghasilkan banyak state anakan, faktor b relatif besar.
- *Space Complexity*: $O(b^d)$ juga karena menyimpan semua frontier dalam memori.
- Kelebihan: Mudah diimplementasikan, menjamin solusi optimal.
- Kekurangan: Memakan waktu dan memori yang banyak karena b^d yang bertumbuh secara cepat dan eksponensial.

3.1.2 Greedy Best First Search

- Frontier

```
prioQueue = new PriorityQueue<State>(Comparator.comparingInt((State s) -> s.heuristic)); // Always process the state with the least cost first
        visited = new HashSet<>();
        nodesExplored = 0;
```

Setiap state akan di-*enqueue* pada sebuah priority queue berdasarkan nilai heuristiknya. Nilai heuristik dapat berupa jarak dari mobil primary ke pintu keluar dalam jumlah sel, atau jumlah mobil yang menghalangi mobil utama ke pintu keluar, atau gabungan dari keduanya.

- Loop Utama
    1. currState: prioQueue.poll(): Mengambil state dengan nilai heuristik terkecil
    2. Goal test: Memeriksa apakah mobil *primary* sudah mencapai sel keluar.
    3. Generate successor: Sama seperti UCS, namun dengan tambahan, saat akan dibuat state baru, state baru akan menerima perhitungan nilai heuristik dari pergerakan tersebut dan menyimpannya. Lalu, state tersebut akan dimasukkan kedalam priority queue.
    4. Lalu state baru yang disimpan akan divalidasi kembali dengan dimasukkan ke dalam set visited.

- Analisis algoritma

- Heuristik: jarak terhadap pintu keluar, jumlah mobil yang menghalangi mobil primer ke pintu keluar, atau keduanya.
- *Completeness*: tidak selalu menemukan solusi optimal karena sifatnya yang menilai sebuah jalur menggunakan nilai heuristik saja. Dapat menghasilkan solusi buntu.
- *Optimality*: Tidak menjamin jalur terpendek
- *Time Complexity*: $O(b^{d'})$ dengan d′ adalah depth hingga solusi ditemukan menggunakan nilai heuristik. Penelusuran yang lebih sedikit dibanding UCS biasanya menghasilkan *output* yang lebih cepat dibanding UCS.
- *Space Complexity*: $O(b^{d'})$ namun lebih hemat dibanding UCS karena alasan yang sama dengan *time complexity*
- Kelebihan: Cepat dalam pencarian jalan keluar, pencarian frontier jauh lebih sedikit apabila heuristik yang dimiliki bagus.
- Kekurangan: Terdapat kemungkinan tidak menemukan solusi, belum tentu menghasilkan solusi optimal, kualitas dari solusi sangat bergantung pada heuristik yang dimilikinya.

3.1.3 A*

- Frontier

```
prioQueue = new PriorityQueue<State>(Comparator.comparingInt((State s) -> s.AScore));
        visited = new HashSet<>();
        nodesExplored = 0;
```

sama seperti GBFS, namun nilai heuristik yang dipakai adalah AScore (gabungan dari nilai heuristik dan biaya penelusuran normal (kedalaman)).

- Loop Utama
    1. currState = prioQueue.poll(): Mengambil state dengan f = g + h terkecil
    2. Goal test: Sama seperti dua algoritma sebelumnya

3. Generate successor: Sama seperti greedy best first search, namun sebelum menambahkan state ke priority queue, state menyimpan hasil perhitungan AScore (f = g + h) yang akan digunakan sebagai *comparator* pada priority queue.
4. Validasi dengan visited.add(movedState).
5.

- Analisis algoritma
    - Heuristik: jarak terhadap pintu keluar, jumlah mobil yang menghalangi mobil primer ke pintu keluar, atau keduanya.
    - *Completenes*s: Selama h(n) *admissible*, algoritma menghasilkan langkah terpendek.
    - *Optimality*: Menghasilkan solusi optimal apabila h(n) *admissible.*
    - *Time complexity*: $O(b^d)$, namun dapat jauh lebih kecil daripada UCS jika heuristic bagus. Namun, jika dibandingkan dengan algoritma GBFS dengan heuristic serupa, kompleksitas waktu A* lebih besar.
    - *Space complexity*: $O(b^d)$, namun dapat jauh lebih kecil daripada UCS jika heuristic bagus. Namun, jika dibandingkan dengan algoritma GBFS dengan heuristic serupa, kompleksitas waktu A* lebih besar.
    - Kelebihan: Menjamin menemukan jalur optimal dan lebih cepat dibanding UCS.
    - Kekurangan: Butuh memori frontier yang masih cukup besar, overhead per‑state lebih tinggi (perbandingan f = g + h), lebih lambat dibanding GBFS.

3.1.4 Pertanyaan

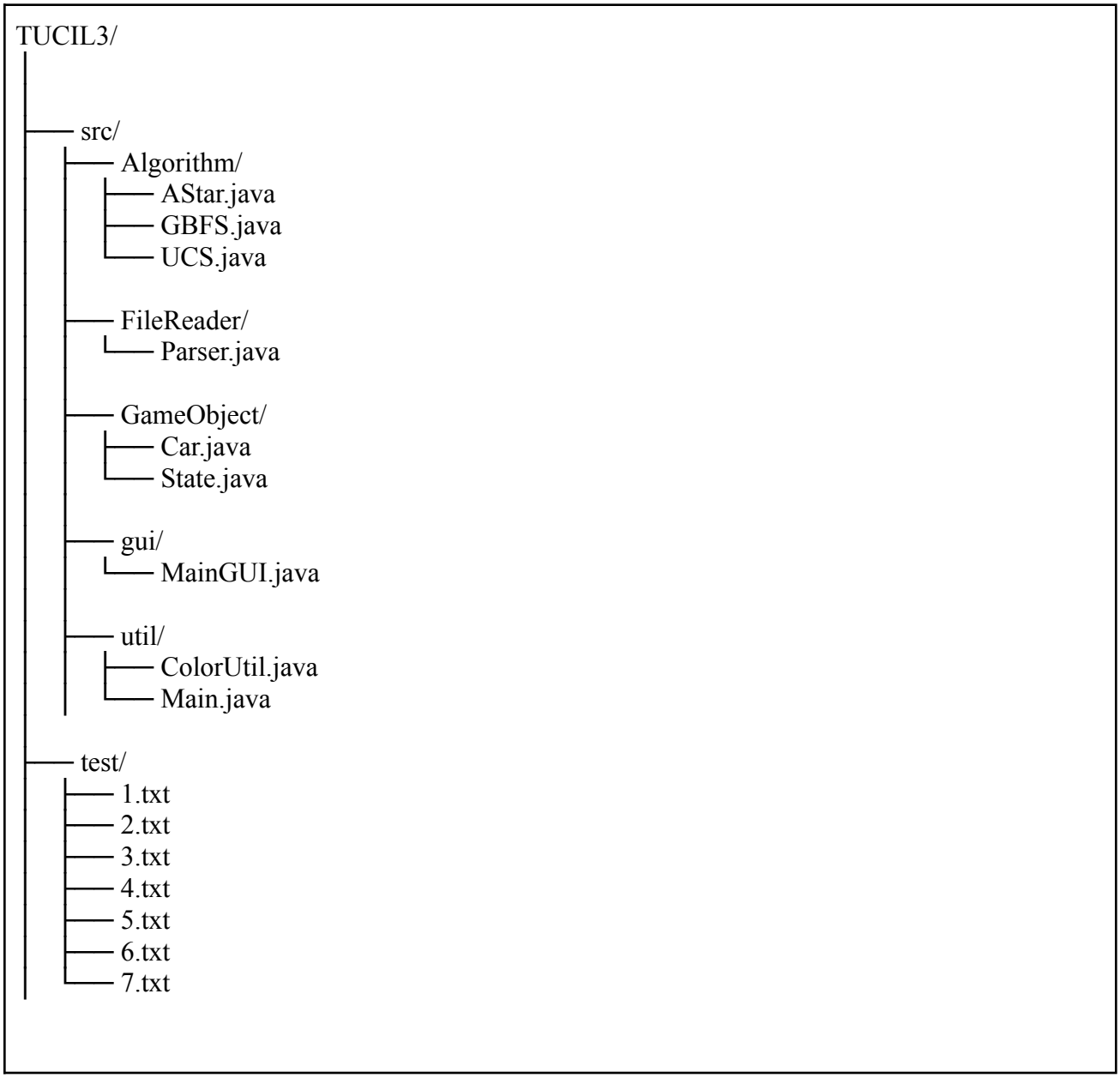
1. Definisi dari f(n) dan g(n), sesuai dengan salindia kuliah:
    - g(n) adalah biaya kumulatif dari simpul awal hingga simpul n.
    - f(n) adalah harga perkiraan dari titik awal hingga titik tujuan
2. Apakah heuristik yang digunakan pada algoritma A* admissible? Jelaskan sesuai definisi admissible dari salindia kuliah:

- Exit-distanc**e** selalu ≤ biaya langkah sebenarnya, karena minimal setiap langkah menggeser mobil utama satu sel menuju pintu. ⇒ admissible.
- Blocker-count mengasumsikan setiap mobil penghalang hanya butuh satu move untuk keluar jalur. Jika ternyata beberapa blocker memerlukan lebih dari satu langkah untuk digeser, blocker-count dapat *overestimate* dan tidak admissible.
- Kombinasi (exit-distance + blocker-count) juga tidak dijamin admissible kecuali blocker-count sudah terbukti tidak melebih-lebihkan

3. Pada penyelesaian Rush Hour, apakah algoritma UCS sama dengan BFS? (dalam artian urutan node yang dibangkitkan dan path yang dihasilkan sama)
    - Hasil: Urutan pengembangan simpul (node expansion order) dan jalur (path) yang dihasilkan sama persis.
    - Kesimpulan: Karena semua edge memiliki cost identik, UCS di kode-mu berperilaku ekivalen dengan BFS.
4. Secara teoritis, apakah algoritma A* lebih efisien dibandingkan dengan algoritma UCS pada penyelesaian Rush Hour?
    - Teori: Jumlah simpul yang dibuka (expanded) oleh A* tidak akan melebihi jumlah simpul yang dibuka UCS, dan biasanya jauh lebih sedikit jika h(n) informatif.
    - Kesimpulan: Secara teoritis, A* lebih efisien daripada UCS pada puzzle Rush Hour, asalkan heuristik memadai.
5. Secara teoritis, apakah algoritma Greedy Best First Search menjamin solusi optimal untuk penyelesaian Rush Hour?
    - Greedy Best-First hanya memprioritaskan simpul ber-h(n) terkecil, tanpa mempertimbangkan g(n)
    - Tanpa komponen cost kumulatif, ia dapat memilih jalur awal yang “terlihat dekat” tetapi memiliki total langkah jauh dari minimal.
    - Kesimpulan: Greedy Best-First tidak menjamin solusi optimal untuk Rush Hour.

3.2 Source Program

Struktur Folder



Algorithm Package

3.2.1. AStar.java

Source Code:

```java
package Algorithm;
import GameObject.*;
import java.util.*;

/*
 * Basically the same as AStar but with the addition for a normal cost
*/

public class AStar {
    private PriorityQueue<State> prioQueue;
    public Set<State> visited;
    public State finalState;
    public int nodesExplored;
    public long time;
    boolean useExitDistH;
    boolean useObsCountH;


    public AStar(int AScoreValue) {
        prioQueue = new PriorityQueue<State>(Comparator.comparingInt((State
s) -> s.AScore)); // Always process the state with the least cost first
        visited = new HashSet<>();
        nodesExplored = 0;
        if (AScoreValue == 0) {
            useExitDistH = true;
        }
        else if (AScoreValue == 1) {
            useObsCountH = true;
        }
        else {
            useExitDistH = true;
            useObsCountH = true;
        }
    }

    public boolean solve(State initialState) {
        long startTime = System.nanoTime();
        System.out.println("generating solution....");
        boolean found = false;
```

```
        prioQueue.add(initialState);

        while (!found) {
        // detect if finish
            State currState = prioQueue.poll();
            if (currState == null) break;
            Car mainCar = currState.cars.get('P'); // retrieve primary car
            if (mainCar.isHorizontal) {
                if (mainCar.col + mainCar.length - 1 == State.exitCol ||
mainCar.col == State.exitCol) {
                    found = true;
                    time = System.nanoTime() - startTime;
                    System.out.println("Exit found");
                    finalState = currState;
                    Car carP = finalState.cars.get('P');
                    carP.col = -1;
                    carP.row = -1;
                    finalState.displayState();
                    break;
                }
            }
            else
            {
                if (mainCar.row + mainCar.length - 1 == State.exitRow ||
mainCar.row == State.exitRow) {
                    found = true;
                    time = System.nanoTime() - startTime;
                    System.out.println("Exit found");
                    finalState = currState;
                    Car carP = finalState.cars.get('P');
                    carP.col = -1;
                    carP.row = -1;
                    finalState.displayState();
                    break;
                }
            }
            // move every car
            char[][] board = currState.buildBoard();
            for (Car car : currState.cars.values()) {
```

```
                if (car.isHorizontal) {
                    int emptyCellsLeft = car.canMoveLeft(board);
                    int emptyCellsRight = car.canMoveRight(board);
                    if (emptyCellsLeft > 0) {
                        for (int i = 0; i < emptyCellsLeft; i++) {
                            State movedState = currState.moveLeft(car.id, i
+ 1);
                            if (visited.add(movedState)) {
                                nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                else if (useExitDistH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
movedState.cost;
                                }
                                else if (useObsCountH) {
                                    movedState.AScore =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                prioQueue.add(movedState);
                            }
                        }
                    }
                    if (emptyCellsRight > 0) {
                        for (int i = 0; i < emptyCellsRight; i++) {
                            State movedState = currState.moveRight(car.id,
i + 1);
                            if (visited.add(movedState)) {
                                nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
```

```
board) + movedState.cost;
                                }
                                else if (useExitDistH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
movedState.cost;
                                }
                                else if (useObsCountH) {
                                    movedState.AScore =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                prioQueue.add(movedState);
                            }
                        }
                    }
                }
                else
                {
                    int emptyCellsTop = car.canMoveUp(board);
                    int emptyCellsBottom = car.canMoveDown(board);
                    if (emptyCellsTop > 0) {
                        for (int i = 0; i < emptyCellsTop; i++) {
                            State movedState = currState.moveUp(car.id, i +
1);
                            if (visited.add(movedState)) {
                                nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                else if (useExitDistH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
movedState.cost;
                                }
                                else if (useObsCountH) {
```

```
                                    movedState.AScore =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                prioQueue.add(movedState);
                            }
                        }
                    }
                    else
                    {
                        if (emptyCellsBottom > 0) {
                            for (int i = 0; i < emptyCellsBottom; i++) {
                                State movedState =
currState.moveDown(car.id, i + 1);
                                if (visited.add(movedState)) {
                                    nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                else if (useExitDistH) {
                                    movedState.AScore =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
movedState.cost;
                                }
                                else if (useObsCountH) {
                                    movedState.AScore =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board) + movedState.cost;
                                }
                                prioQueue.add(movedState);
                                }
                            }
                        }
                    }
                }
            }
```

```
        }
        return found;
    }

    public int getExitDist(Car car, int exitRow, int exitCol) {
        int dist = 0; // in cells
        if (car.isHorizontal) {
            if (exitRow >= car.row + car.length - 1) {
                dist = exitRow - (car.row + car.length - 1);
            }
            else {
                dist = car.row - exitRow;
            }
        }
        else {
            if (exitCol >= car.col + car.length - 1) {
                dist = exitCol - (car.col + car.length - 1);
            }
            else {
                dist = car.col - exitCol;
            }
        }
        return dist;
    }

    public int getBlockersAmount(Car car, int exitRow, int exitCol,
char[][] board) {
        int amount = 0; // amount of cars blocking the exit from the
primary car

        Set<Character> blockers = new HashSet<>(); // set to car or
character?

        if (car.isHorizontal) {
            if (exitCol >= car.col + car.length - 1) { // if exit on right
                for (int i = car.col + car.length; i <= exitRow; i++) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
```

```
                }
            }
            else {
                for (int i = car.col - 1; i >= exitCol; i--) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }

        }
        else
        {
            if (exitRow >= car.row + car.length - 1) { // if exit on right
                for (int i = car.row + car.length; i <= exitRow; i++) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }
            else {
                for (int i = car.row - 1; i >= exitRow; i--) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }
        }
        return amount;
    }

    public double getRuntime() {
        return time / 1_000_000.0; // Convert nanoseconds to milliseconds
    }
}
```

Nama	Jenis	Deskripsi	Return
AStar	Class	Implementasi algoritma A* untuk pathfinding, menggabungkan cost dan heuristik.	—
prioQueue	Field	Priority queue dari state, diurutkan berdasarkan AScore (fungsi f(n)).	—
visited	Field	Set of visited states untuk menghindari eksplorasi ulang.	—
finalState	Field	State ketika mobil utama mencapai pintu keluar.	—
nodesExplored	Field	Jumlah state yang telah dieksplorasi selama pencarian.	—
time	Field	Lama waktu eksekusi algoritma dalam nanodetik.	—
useExitDistH	Field	Apakah heuristik jarak ke pintu keluar digunakan.	—
useObsCountH	Field	Apakah heuristik jumlah penghalang digunakan.	—
AStar(int AScoreValue)	Constructor	Konstruktor untuk menentukan jenis heuristik yang digunakan (0, 1, atau kombinasi).	—
solve(State initialState)	Method	Menjalankan algoritma A* dari state awal dan mencari jalur ke pintu keluar.	boolean (true jika solusi ditemukan)
getExitDist(Car car, int exitRow, int exitCol)	Method	Menghitung jarak dari mobil utama ke pintu keluar sebagai heuristik.	int (jarak dalam sel)
getBlockersAmount(Car car, int exitRow, int exitCol, char[][] board)	Method	Menghitung jumlah mobil lain yang menghalangi jalan keluar.	int (jumlah blocker)

getRuntime()	Method	Mengonversi dan mengembalikan waktu pencarian dari nanodetik ke milidetik.	double (ms)

3.2.2. GBFS.java

```
package Algorithm;
import GameObject.*;
import java.util.*;

/*
 * Greedy Best First Search Algorithm
 * heuristic: The closer the primary car is to the exit door (measured in
cell between primary car and the exit)
 */

 /*
  * Special cases:
  1. Car right next to exit
  2.
  */

  /*
   * General Idea:
   * 1. Init:
   *    - recieve a current state of a board
   *    - initiate cost:
   *        1) Get distance between primary car and exit
(getExitDistance())
   *        2) Cost = getExitDistance() + getBlockers
   * 2. Final state: If getExitDistance() = 0
   * 3. Loop
   *    You have a state. Each state has a map of cars. For each car that`
   *
   * Other notes:
   * Use set to know visited state
   * prio queue for gbfs (Use BFS technique for the algorithm)
   * Car is horizontal or vertical
```

```
   */

public class GBFS {
    private PriorityQueue<State> prioQueue;
    public Set<State> visited;
    public State finalState;
    public int nodesExplored;
    public long time;
    boolean useExitDistH;
    boolean useObsCountH;


    public GBFS(int heuristicValue) {
        prioQueue = new PriorityQueue<State>(Comparator.comparingInt((State
s) -> s.heuristic)); // Always process the state with the least cost first
        visited = new HashSet<>();
        nodesExplored = 0;
        if (heuristicValue == 0) {
            useExitDistH = true;
        }
        else if (heuristicValue == 1) {
            useObsCountH = true;
        }
        else {
            useExitDistH = true;
            useObsCountH = true;
        }
    }

    public boolean solve(State initialState) {
        long startTime = System.nanoTime();
        System.out.println("generating solution....");
        boolean found = false;
        prioQueue.add(initialState);

        while (!found) {
        // detect if finish
            State currState = prioQueue.poll();
            if (currState == null) break;
```

```
            Car mainCar = currState.cars.get('P'); // retrieve primary car
            if (mainCar.isHorizontal) {
                if (mainCar.col + mainCar.length - 1 == State.exitCol ||
mainCar.col == State.exitCol) {
                    found = true;
                    time = System.nanoTime() - startTime;
                    System.out.println("Exit found");
                    finalState = currState;
                    Car carP = finalState.cars.get('P');
                    carP.col = -1;
                    carP.row = -1;
                    finalState.displayState();
                    break;
                }
            }
            else
            {
                if (mainCar.row + mainCar.length - 1 == State.exitRow ||
mainCar.row == State.exitRow) {
                    found = true;
                    time = System.nanoTime() - startTime;
                    System.out.println("Exit found");
                    finalState = currState;
                    Car carP = finalState.cars.get('P');
                    carP.col = -1;
                    carP.row = -1;
                    finalState.displayState();
                    break;
                }
            }
            // move every car
            char[][] board = currState.buildBoard();
            for (Car car : currState.cars.values()) {
                if (car.isHorizontal) {
                    int emptyCellsLeft = car.canMoveLeft(board);
                    int emptyCellsRight = car.canMoveRight(board);
                    if (emptyCellsLeft > 0) {
                        for (int i = 0; i < emptyCellsLeft; i++) {
                            State movedState = currState.moveLeft(car.id, i
```

```
+ 1);
                            if (visited.add(movedState)) {
                                nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                }
                                else if (useExitDistH) {
                                    movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol);
                                }
                                else if (useObsCountH) {
                                    movedState.heuristic =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                }
                                prioQueue.add(movedState);
                            }
                        }
                    }
                    if (emptyCellsRight > 0) {
                        for (int i = 0; i < emptyCellsRight; i++) {
                            State movedState = currState.moveRight(car.id,
i + 1);
                            if (visited.add(movedState)) {
                                nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                }
                                else if (useExitDistH) {
                                    movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol);
                                }
                                else if (useObsCountH) {
```

```
                                    movedState.heuristic =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                }
                                prioQueue.add(movedState);
                            }
                        }
                    }
                    else
                    {
                        int emptyCellsTop = car.canMoveUp(board);
                        int emptyCellsBottom = car.canMoveDown(board);
                        if (emptyCellsTop > 0) {
                            for (int i = 0; i < emptyCellsTop; i++) {
                                State movedState = currState.moveUp(car.id, i +
1);
                                if (visited.add(movedState)) {
                                    nodesExplored++;
                                    if (useExitDistH && useObsCountH) {
                                        movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                    }
                                    else if (useExitDistH) {
                                        movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol);
                                    }
                                    else if (useObsCountH) {
                                        movedState.heuristic =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                    }
                                    prioQueue.add(movedState);
                                }
                            }
                        }
                        else
```

```
                    {
                        if (emptyCellsBottom > 0) {
                            for (int i = 0; i < emptyCellsBottom; i++) {
                                State movedState =
currrState.moveDown(car.id, i + 1);
                                if (visited.add(movedState)) {
                                    nodesExplored++;
                                if (useExitDistH && useObsCountH) {
                                    movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol) +
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                }
                                else if (useExitDistH) {
                                    movedState.heuristic =
getExitDist(movedState.cars.get('P'), State.exitRow, State.exitCol);
                                }
                                else if (useObsCountH) {
                                    movedState.heuristic =
getBlockersAmount(movedState.cars.get('P'), State.exitRow, State.exitCol,
board);
                                }
                                prioQueue.add(movedState);
                                }
                            }
                        }
                    }
                }
            }
        }
        return found;
    }

    public int getExitDist(Car car, int exitRow, int exitCol) {
        int dist = 0; // in cells
        if (car.isHorizontal) {
            if (exitRow >= car.row + car.length - 1) {
                dist = exitRow - (car.row + car.length - 1);
            }
```

```
            else {
                dist = car.row - exitRow;
            }
        }
        else {
            if (exitCol >= car.col + car.length - 1) {
                dist = exitCol - (car.col + car.length - 1);
            }
            else {
                dist = car.col - exitCol;
            }
        }
        return dist;
    }

    public int getBlockersAmount(Car car, int exitRow, int exitCol,
char[][] board) {
        int amount = 0; // amount of cars blocking the exit from the
primary car

        Set<Character> blockers = new HashSet<>(); // set to car or
character?

        if (car.isHorizontal) {
            if (exitCol >= car.col + car.length - 1) { // if exit on right
                for (int i = car.col + car.length; i <= exitRow; i++) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }
            else {
                for (int i = car.col - 1; i >= exitCol; i--) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }
```

```
        }
        else
        {
            if (exitRow >= car.row + car.length - 1) { // if exit on right
                for (int i = car.row + car.length; i <= exitRow; i++) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }
            else {
                for (int i = car.row - 1; i >= exitRow; i--) {
                    if (board[car.row][i] != '.' && blockers.add(car.id)) {
                        amount++;
                    }
                }
            }
        }
        return amount;
    }

    public double getRuntime() {
        return time / 1_000_000.0; // Convert nanoseconds to milliseconds
    }
}
```

Nama	Jenis	Deskripsi	Return
GBFS	Class	Kelas yang mengimplementasikan algoritma Greedy Best First Search untuk menyelesaikan puzzle Rush Hour.	—
prioQueue	Field	PriorityQueue yang menyimpan State, diprioritaskan berdasarkan nilai heuristic.	—

visited	Field	Set untuk menyimpan state yang sudah dikunjungi agar tidak diulang.	—
finalState	Field	State akhir ketika mobil utama mencapai pintu keluar.	—
nodesExplored	Field	Jumlah total node yang telah dieksplorasi selama pencarian.	—
time	Field	Waktu total eksekusi pencarian dalam satuan nanodetik.	—
useExitDistH	Field	Flag yang menandakan apakah heuristik berdasarkan jarak ke pintu digunakan.	—
useObsCountH	Field	Flag yang menandakan apakah heuristik berdasarkan jumlah penghalang digunakan.	—
GBFS(int heuristicValue)	Constructor	Konstruktor yang menginisialisasi algoritma dengan pilihan heuristik (0, 1, atau kombinasi keduanya).	—
boolean solve(State initialState)	Method	Menjalankan algoritma GBFS dari initialState. Mencari solusi hingga primary car mencapai pintu keluar.	boolean
int getExitDist(Car car, int row, int col)	Method	Mengembalikan estimasi jarak antara mobil utama dan pintu keluar (heuristik 1).	int
int getBlockersAmount(Car car, int row, int col, char[][] board)	Method	Menghitung jumlah mobil yang menghalangi jalan mobil utama ke pintu keluar (heuristik 2).	int
double getRuntime()	Method	Mengonversi dan mengembalikan waktu pencarian dalam satuan milidetik.	double

3.2.3. UCS.java

```
package Algorithm;

import GameObject.*;
```

```
import java.util.*;

public class UCS {
    public Queue<State> queue;
    public Set<State> visited;
    public State finalState;
    public int nodesExplored;
    public long time;

    public UCS(){
        queue = new LinkedList<>();
        visited = new HashSet<>();
        nodesExplored = 0;
    }

    public boolean solve(State initialState) {
        long startTime = System.nanoTime();
        System.out.println("generating solution....");
        boolean found = false;
        queue.add(initialState);
        while (!queue.isEmpty() && !found) {
            State currentState = queue.poll();
            Car mainCar = currentState.cars.get('P');
            if (mainCar.isHorizontal){
                if(mainCar.col<=State.exitCol && State.exitCol
<=mainCar.col+mainCar.length-1){
                    found = true;
                    time = System.nanoTime() - startTime;
                    System.out.println("found exit");
                    finalState = currentState;
                    Car carP = currentState.cars.get('P');
                    carP.col = -1;
                    carP.row = -1;
                    currentState.displayState();
                    break;
                }
            }
            else{
                if(mainCar.row<=State.exitRow && State.exitRow
```

```
<=mainCar.row+mainCar.length-1){
                    found = true;
                    time = System.nanoTime() - startTime;
                    System.out.println("found exit");
                    finalState = currentState;
                    Car carP = currentState.cars.get('P');
                    carP.col = -1;
                    carP.row = -1;
                    currentState.displayState();
                    break;
                }
            }
            char[][] board = currentState.buildBoard();
            for(Car car: currentState.cars.values()){
                if(car.isHorizontal){
                    int moveLeft = car.canMoveLeft(board);
                    int moveRight = car.canMoveRight(board);
                    if(moveLeft>0){
                        for (int i=0;i<moveLeft;i++){
                            State moveState = currentState.moveLeft(car.id,
i+1);
                            if (visited.add(moveState)) {
                                nodesExplored++;
                                queue.add(moveState);
                            }
                        }
                    }
                    if(moveRight>0){
                        for (int i = 0; i < moveRight; i++) {
                            State moveState =
currentState.moveRight(car.id, i+1);
                            if (visited.add(moveState)) {
                                nodesExplored++;
                                queue.add(moveState);
                            }
                        }
                    }
                }
                else{
```

```
                    int moveUp = car.canMoveUp(board);
                    int moveDown = car.canMoveDown(board);
                    if (moveUp>0){
                        for (int i = 0; i < moveUp; i++) {
                            State moveState = currentState.moveUp(car.id,
i+1);
                            if (visited.add(moveState)) {
                                nodesExplored++;
                                queue.add(moveState);
                            }
                        }
                    }
                    if (moveDown>0){
                        for (int i = 0; i < moveDown; i++) {
                            State moveState = currentState.moveDown(car.id,
i+1);
                            if (visited.add(moveState)) {
                                nodesExplored++;
                                queue.add(moveState);
                            }
                        }
                    }
                }
            }
        }
        if(!found){
            time = System.nanoTime() - startTime;
        }
        return found;
    }

    public double getRuntime() {
        return time / 1_000_000.0; // Convert nanoseconds to milliseconds
    }

}
```

Nama	Jenis	Deskripsi	Return
UCS	Class	Implementasi algoritma Uniform Cost Search (UCS) untuk menyelesaikan puzzle Rush Hour.	—
queue	Field	Queue (FIFO) dari State yang menyimpan urutan state yang akan dieksplorasi.	—
visited	Field	Set untuk menyimpan state yang sudah dikunjungi agar tidak dieksplorasi ulang.	—
finalState	Field	State akhir saat mobil utama berhasil mencapai pintu keluar.	—
nodesExplored	Field	Jumlah total node/state yang telah dieksplorasi.	—
time	Field	Waktu total eksekusi algoritma dalam satuan nanodetik.	—
UCS()	Constructor	Konstruktor untuk inisialisasi struktur queue dan visited set, serta reset jumlah eksplorasi.	—
boolean solve(State initialState)	Method	Menjalankan algoritma UCS dari initialState, mengeksplorasi hingga menemukan jalan ke pintu keluar.	boolean (true jika solusi ditemukan)
double getRuntime()	Method	Mengembalikan waktu eksekusi pencarian dalam satuan milidetik.	double

FileReader Package

3.2.4. Parser.java

```
package FileReader;

import GameObject.Car;
import GameObject.State;
```

```
import java.io.*;
import java.util.*;

public class Parser {
    private static int clamp(int v, int min, int max) {
        return Math.max(min, Math.min(max, v));
    }

    public static State loadState(String filename) throws IOException {
        File file = new File(filename);
        if (!file.exists() || file.isDirectory()) {
            file = new File("test" + File.separator + filename);
            if (!file.exists() || file.isDirectory()) {
                throw new FileNotFoundException("File not found: " +
filename);
            }
        }
        BufferedReader reader = new BufferedReader(new FileReader(file));

        String line = reader.readLine();
        if (line == null)
            throw new IllegalArgumentException("File kosong");
        String[] dims = line.trim().split("\\s+");
        if (dims.length != 2)
            throw new IllegalArgumentException("Baris pertama harus dua
angka: height width");
        int height = Integer.parseInt(dims[0]);
        int width = Integer.parseInt(dims[1]);

        line = reader.readLine();
        if (line == null)
            throw new IllegalArgumentException("File kurang baris untuk
piece count");
        int pieceCount = Integer.parseInt(line.trim());
        if (pieceCount < 1)
            throw new IllegalArgumentException("Jumlah pieces harus >= 1");

        List<String> lines = new ArrayList<>();
```

```
        while ((line = reader.readLine()) != null) {
            lines.add(line);
        }
        reader.close();
        if (lines.size() < height)
            throw new IllegalArgumentException(
                    "Baris papan kurang: diharapkan " + height + ", tapi
file punya " + lines.size());

        int boardStart = -1;
        for (int i = 0; i < lines.size(); i++) {
            if (lines.get(i).stripLeading().length() >= width) {
                boardStart = i;
                break;
            }
        }
        if (boardStart < 0 || boardStart + height > lines.size())
            throw new IllegalArgumentException("Tidak ditemukan blok papan
yang valid");

        char[][] rawBoard = new char[height][width];
        boolean[][] occupied = new boolean[height][width];
        int exitRow = -1, exitCol = -1;
        boolean foundP = false;

        for (int r = 0; r < height; r++) {
            String rawLine = lines.get(boardStart + r);
            String trimmed = rawLine.stripLeading();
            int tlen = trimmed.length();
            String rowData;
            // Case: exact width
            if (tlen == width) {
                rowData = trimmed;
            }
            // Case: K at left outside
            else if (tlen == width + 1 && trimmed.charAt(0) == 'K') {
                exitRow = r;
                exitCol = 0;
                rowData = trimmed.substring(1);
```

```
            }
            // Case: K at right outside
            else if (tlen == width + 1 && trimmed.charAt(tlen - 1) == 'K')
{
                exitRow = r;
                exitCol = width - 1;
                rowData = trimmed.substring(0, width);
            } else {
                throw new IllegalArgumentException(
                        "Baris ke-" + r + " harus panjang " + width + "
atau " + (width + 1)
                                + ", tapi panjangnya " + tlen);
            }

            // Fill board row
            for (int c = 0; c < width; c++) {
                char ch = rowData.charAt(c);
                rawBoard[r][c] = ch;
                if (ch == 'P')
                    foundP = true;
                if (ch != '.') {
                    if (occupied[r][c])
                        throw new IllegalArgumentException(
                                "Overlap pada piece di posisi (" + r + ","
+ c + ")");
                    occupied[r][c] = true;
                }
            }
        }

        int indentBoard = lines.get(boardStart).length() -
lines.get(boardStart).stripLeading().length();
        for (int i = 0; i < lines.size(); i++) {
            String rawLine2 = lines.get(i);
            for (int k = 0; k < rawLine2.length(); k++) {
                if (rawLine2.charAt(k) == 'K') {
                    if (i < boardStart) {
                        exitRow = 0;
                    }
```

```
                    else if (i >= boardStart + height) {
                        exitRow = height - 1;
                    }
                    int relC = clamp(k - indentBoard, 0, width - 1);
                    exitCol = relC;
                }
            }
        }

        if (exitRow < 0)
            throw new IllegalArgumentException("Tidak ditemukan exit 'K' di
file");

        if (!foundP)
            throw new IllegalArgumentException("Harus ada main car 'P' di
papan");

        Map<Character, List<int[]>> positions = new HashMap<>();
        for (int r = 0; r < height; r++) {
            for (int c = 0; c < width; c++) {
                char ch = rawBoard[r][c];
                if (ch != '.') {
                    positions.computeIfAbsent(ch, k -> new
ArrayList<>()).add(new int[] { r, c });
                }
            }
        }
        int distinct = positions.size() - (positions.containsKey('P') ? 1 :
0);
        if (distinct != pieceCount)
            throw new IllegalArgumentException(
                    "Jumlah pieces tidak sesuai, diharapkan " + pieceCount
+ ", tapi ditemukan " + distinct);

        State.initBoard(height, width, exitRow, exitCol);
        State state = new State(null, ' ', 0, 0);
        for (Map.Entry<Character, List<int[]>> e : positions.entrySet()) {
            char id = e.getKey();
            List<int[]> pts = e.getValue();
```

```
            int minR = pts.stream().mapToInt(p -> p[0]).min().getAsInt();
            int minC = pts.stream().mapToInt(p -> p[1]).min().getAsInt();
            boolean horiz = pts.stream().allMatch(p -> p[0] == minR);
            int len = pts.size();
            boolean main = (id == 'P');
            Car car = new Car(minR, minC, horiz, len, id, main);
            state.cars.put(id, car);
        }
        return state;
    }
}
```

Nama	Jenis	Deskripsi	Return
Parser	Class	Utilitas untuk membaca file konfigurasi puzzle Rush Hour dan mengubahnya menjadi objek State.	—
clamp(int v, int min, int max)	Method (private static)	Membatasi nilai v agar tetap berada dalam rentang [min, max].	int
loadState(String filename)	Method (public static)	Membaca file teks puzzle, mengecek validitas, mendeteksi posisi exit K, menyusun mobil, dan mengembalikan objek State.	State

GameObject Package

3.2.5. Car.java

```
package GameObject;

import java.util.Objects;

public class Car {
    public int row, col;
```

```
    public boolean isHorizontal;
    public int length;
    public char id;
    public boolean isMainCar;

    public Car(int row, int col, boolean isHorizontal, int length, char id,
boolean isMainCar) {
        this.row = row;
        this.col = col;

        this.isHorizontal = isHorizontal;
        this.length = length;
        this.id = id;
        this.isMainCar = isMainCar;
    }

    public Car(Car other){
        this.row = other.row;
        this.col = other.col;
        this.isHorizontal = other.isHorizontal;
        this.length = other.length;
        this.id = other.id;
        this.isMainCar = other.isMainCar;
    }

    //integer means the number of steps that can be moved
    public int canMoveRight(char[][] board) {
        int steps = 0;
        for (int i = col+length-1; i < board[0].length ; i++){
            if (board[row][i] == '.'){
                steps++;
            }
            else if (board[row][i] == id) {
                continue;
            }
            else {
                break;
            }
        }
```

```
        return steps;
    }

    public int canMoveLeft(char[][] board) {
        int steps = 0;
        for (int i = col; i >= 0; i--) {
            if (board[row][i] == '.') {
                steps++;
            }
            else if (board[row][i] == id) {
                continue;
            }
            else {
                break;
            }
        }
        return steps;
    }

    public int canMoveUp(char[][] board) {
        int steps = 0;
        for (int i = row; i >= 0; i--) {
            if (board[i][col] == '.') {
                steps++;
            }
            else if (board[i][col] == id) {
                continue;
            }
            else {
                break;
            }
        }
        return steps;
    }

    public int canMoveDown(char[][] board) {
        int steps = 0;

        for (int i = row+length-1; i < board.length; i++) {
```

```
            if (board[i][col] == '.') {
                steps++;
            }
            else if (board[i][col] == id) {
                continue;
            }
            else {
                break;
            }
        }
        return steps;
    }

    public Car moveRight(int steps) {
        return new Car(row, col + steps, isHorizontal, length, id,
isMainCar);
    }

    public Car moveLeft(int steps) {
        return new Car(row, col - steps, isHorizontal, length, id,
isMainCar);
    }

    public Car moveUp(int steps) {
        return new Car(row - steps, col, isHorizontal, length, id,
isMainCar);
    }

    public Car moveDown(int steps) {
        return new Car(row + steps, col, isHorizontal, length, id,
isMainCar);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Car)) return false;
        Car other = (Car) obj;
        return row == other.row && col == other.col && isHorizontal ==
```

```
other.isHorizontal && length == other.length && id == other.id;
    }

    @Override
    public int hashCode() {
        return Objects.hash(row, col, isHorizontal, length, id);
    }
}
```

Nama	Jenis	Deskripsi	Return
Car	Class	Mewakili sebuah mobil dalam puzzle Rush Hour dengan posisi, orientasi, panjang, dan ID unik.	—
row	Field (public)	Posisi baris awal mobil (koordinat kiri-atas).	int
col	Field (public)	Posisi kolom awal mobil (koordinat kiri-atas).	int
isHorizontal	Field (public)	Menyatakan orientasi mobil: true jika horizontal, false jika vertikal.	boolean
length	Field (public)	Panjang mobil dalam satuan sel.	int
id	Field (public)	Karakter ID unik untuk membedakan mobil.	char
isMainCar	Field (public)	Menyatakan apakah mobil ini adalah mobil utama P.	boolean
Car(int, int, boolean, int, char, boolean)	Constructor	Inisialisasi mobil dengan atribut lengkap: posisi, arah, panjang, ID, dan status utama.	—
Car(Car other)	Constructor (copy)	Membuat salinan mobil dari objek Car lain.	—

int canMoveRight(char[][] board)	Method (public)	Mengembalikan jumlah langkah yang bisa dilakukan ke kanan pada papan.	int
int canMoveLeft(char[][] board)	Method (public)	Mengembalikan jumlah langkah yang bisa dilakukan ke kiri pada papan.	int
int canMoveUp(char[][] board)	Method (public)	Mengembalikan jumlah langkah yang bisa dilakukan ke atas pada papan.	int
int canMoveDown(char[][] board)	Method (public)	Mengembalikan jumlah langkah yang bisa dilakukan ke bawah pada papan.	int
Car moveRight(int steps)	Method (public)	Mengembalikan mobil baru yang telah digeser ke kanan sebanyak steps.	Car
Car moveLeft(int steps)	Method (public)	Mengembalikan mobil baru yang telah digeser ke kiri sebanyak steps.	Car
Car moveUp(int steps)	Method (public)	Mengembalikan mobil baru yang telah digeser ke atas sebanyak steps.	Car
Car moveDown(int steps)	Method (public)	Mengembalikan mobil baru yang telah digeser ke bawah sebanyak steps.	Car
boolean equals(Object obj)	Override Method	Membandingkan dua mobil berdasarkan posisi, arah, panjang, dan ID. Digunakan untuk hashing dan set.	boolean
int hashCode()	Override Method	Menghasilkan nilai hash dari mobil, mendukung penyimpanan dalam struktur seperti HashSet.	int

3.2.6. State.java

```java
package GameObject;
import java.util.*;
import util.ColorUtil;

public class State {
    public Map<Character,Car> cars;
    public State parent;
    public char carId;
    public int direction; // 0=left, 1=right, 2=up, 3=down.
    public int cost;
```

```
    public int heuristic;
    public int AScore;

    //board size
    public static int height;
    public static int width;
    public static int exitRow; //exit row
    public static int exitCol; //exit column

    public State(State parent, char carId, int direction, int cost) {
        cars = new HashMap<Character,Car>();
        this.parent = parent;
        this.carId = carId;
        this.direction = direction;
        this.cost = cost;
    }

    public State(State parent, char carId, int direction, int cost, int
heuristic) {
        cars = new HashMap<Character, Car>();
        this.parent = parent;
        this.carId = carId;
        this.direction = direction;
        this.cost = cost;
        this.heuristic = heuristic;
    }

    public State(State parent, char carId, int direction, int cost, int
heuristic, int AScore) {
        cars = new HashMap<Character, Car>();
        this.parent = parent;
        this.carId = carId;
        this.direction = direction;
        this.cost = cost;
        this.heuristic = heuristic;
        this.AScore = AScore;
    }

    public static void initBoard(int h, int w, int r, int c) {
```

```
        height = h;
        width = w;
        exitRow = r;
        exitCol = c;
    }

    public char[][] buildBoard() {
        char[][] board = new char[height][width];
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                board[i][j] = '.';
            }
        }
        for (Car car : cars.values()) {
            if(car.col < 0 || car.col >= width || car.row < 0 || car.row >=
height) {
                continue; // Skip cars that are out of bounds
            }
            if (car.isHorizontal) {
                for (int i = 0; i < car.length; i++) {
                    board[car.row][car.col + i] = car.id;
                }
            } else {
                for (int i = 0; i < car.length; i++) {
                    board[car.row + i][car.col] = car.id;
                }
            }
        }
        return board;
    }

    public void printBoard() {
        char[][] board = buildBoard();
        Car mainCar = cars.get('P');
        char[][] newBoard;
        if (mainCar.isHorizontal){
            newBoard = new char[height][width+1];
            if (exitCol == 0){
                for (int i = 0; i < height; i++) {
```

```
                for (int j = 0; j < width+1; j++) {
                    if (i == exitRow && j == exitCol) {
                        newBoard[i][j] = 'K';
                    } else if (j == 0) {
                        newBoard[i][j] = ' ';
                    }
                    else {
                        newBoard[i][j] = board[i][j-1];
                    }
                }
            }
        }
        else{
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width + 1; j++) {
                    if (i == exitRow && j == exitCol+1) {
                        newBoard[i][j] = 'K';
                    } else if (j == width) {
                        newBoard[i][j] = ' ';
                    } else {
                        newBoard[i][j] = board[i][j];
                    }
                }
            }
        }
    }
    else{
        newBoard = new char[height+1][width];
        if (exitRow == 0) {
            for (int i = 0; i < height+1; i++) {
                for (int j = 0; j < width; j++) {
                    if (i == exitRow && j == exitCol) {
                        newBoard[i][j] = 'K';
                    } else if (i == 0) {
                        newBoard[i][j] = ' ';
                    } else {
                        newBoard[i][j] = board[i-1][j];
                    }
                }
```

```
                }
            } else {
                for (int i = 0; i < height+1; i++) {
                    for (int j = 0; j < width; j++) {
                        if (i == exitRow+1 && j == exitCol) {
                            newBoard[i][j] = 'K';
                        } else if (i == height) {
                            newBoard[i][j] = ' ';
                        } else {
                            newBoard[i][j] = board[i][j];
                        }
                    }
                }
            }
        }
        for (int i = 0; i < newBoard.length; i++) {
            for (int j = 0; j < newBoard[0].length; j++) {
                if (newBoard[i][j]==carId){
                    System.out.print(ColorUtil.colorize(newBoard[i][j],
ColorUtil.YELLOW) + " ");
                }
                else{
                    System.out.print(newBoard[i][j] + " ");
                }

            }
            System.out.println();
        }
        System.out.println("--------------------------");


    }

    public State moveLeft(char carId, int steps) {
        Car car = cars.get(carId);
        if (car != null) {
            Car newCar = car.moveLeft(steps);
            State newState = new State(this, carId, 0, this.cost+1);
            newState.cars.put(carId, newCar);
```

```
            for (Car car2 : cars.values()) {
                if (car2.id != carId) {
                    newState.cars.put(car2.id, car2);
                }
            }
            return newState;
        }
        return null;
    }

    public State moveRight(char carId, int steps) {
        Car car = cars.get(carId);
        if (car != null) {
            Car newCar = car.moveRight(steps);
            State newState = new State(this, carId, 1, this.cost+1);
            newState.cars.put(carId, newCar);
            for (Car car2 : cars.values()) {
                if (car2.id != carId) {
                    newState.cars.put(car2.id, car2);
                }
            }
            return newState;
        }
        return null;
    }

    public State moveUp(char carId, int steps) {
        Car car = cars.get(carId);
        if (car != null) {
            Car newCar = car.moveUp(steps);
            State newState = new State(this, carId, 2, this.cost + 1);
            newState.cars.put(carId, newCar);
            for (Car car2 : cars.values()) {
                if (car2.id != carId) {
                    newState.cars.put(car2.id, car2);
                }
            }
            return newState;
        }
```

```
        return null;
    }

    public State moveDown(char carId, int steps) {
        Car car = cars.get(carId);
        if (car != null) {
            Car newCar = car.moveDown(steps);
            State newState = new State(this, carId, 3, this.cost + 1);
            newState.cars.put(carId, newCar);
            for (Car car2 : cars.values()) {
                if (car2.id != carId) {
                    newState.cars.put(car2.id, car2);
                }
            }
            return newState;
        }
        return null;
    }

    public void displayState() {
        if (parent != null) {
            parent.displayState();
            if (direction == 0) {
                System.out.println("Move car " + carId + " to the left");
            } else if (direction == 1) {
                System.out.println("Move car " + carId + " to the right");
            } else if (direction == 2) {
                System.out.println("Move car " + carId + " up");
            } else if (direction == 3) {
                System.out.println("Move car " + carId + " down");
            }
            this.printBoard();
        }
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        State other = (State) obj;
```

```
        for (Map.Entry<Character, Car> entry : cars.entrySet()) {
            char carId = entry.getKey();
            Car car1 = entry.getValue();
            Car car2 = other.cars.get(carId);
            if (car2 == null || !car1.equals(car2)) {
                return false;
            }
        }
        return true;
    }

    @Override
    public int hashCode() {
        return Objects.hash(cars);
    }


}
```

Nama	Jenis	Deskripsi	Return
State	Class	Mewakili sebuah state papan permainan yang berisi posisi semua mobil dan status pergerakan.	—
cars	Field (public)	Map yang menyimpan mobil-mobil dengan key karakter ID dan value objek Car.	Map<Character,Car>
parent	Field (public)	State sebelumnya (parent) untuk melacak jejak solusi.	State
carId	Field (public)	ID mobil yang terakhir digerakkan.	char
direction	Field (public)	Arah gerakan terakhir: 0=left, 1=right, 2=up, 3=down.	int

cost	Field (public)	Biaya (jumlah langkah) kumulatif dari awal hingga state ini.	int
heuristic	Field (public)	Nilai heuristic (untuk GBFS).	int
AScore	Field (public)	Nilai total (cost + heuristic) digunakan pada A*.	int
height, width, exitRow, exitCol	Field (public static)	Ukuran papan permainan dan lokasi pintu keluar.	int
State(...)	Constructor	Tiga overloaded constructor untuk membuat state dengan parameter yang berbeda: cost saja, +heuristik, +AScore.	—
initBoard(int h, int w, int r, int c)	Static Method	Mengatur parameter global tinggi, lebar, dan posisi exit pada papan.	void
buildBoard()	Method	Mengembalikan representasi papan 2D dari posisi mobil-mobil saat ini.	char[][]
printBoard()	Method	Menampilkan papan permainan ke console, termasuk posisi mobil dan pintu keluar (K).	void
moveLeft(char carId, int steps)	Method	Mengembalikan state baru jika mobil carId digerakkan ke kiri sejumlah steps.	State
moveRight(char carId, int steps)	Method	Mengembalikan state baru jika mobil carId digerakkan ke kanan sejumlah steps.	State
moveUp(char carId, int steps)	Method	Mengembalikan state baru jika mobil carId digerakkan ke atas sejumlah steps.	State
moveDown(char carId, int steps)	Method	Mengembalikan state baru jika mobil carId digerakkan ke bawah sejumlah steps.	State

displayState()	Method	Menampilkan langkah-langkah dari awal hingga state ini, serta board setiap langkah.	void
equals(Object obj)	Override Method	Membandingkan apakah dua state memiliki posisi mobil yang sama.	boolean
hashCode()	Override Method	Menghasilkan hash unik berdasarkan posisi semua mobil di state ini.	int

GUI Package

3.2.7. MainGUI.java

```
package gui;

import FileReader.Parser;
import Algorithm.AStar;
import Algorithm.GBFS;
import Algorithm.UCS;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.*;
import java.util.List;

import GameObject.Car;
import GameObject.State;

public class MainGUI extends JFrame {
    private static final Color BACKGROUND_COLOR = new Color(240, 240, 245);
    private static final Color BOARD_COLOR = new Color(220, 220, 225);
```

```
    private static final Color EXIT_COLOR = new Color(255, 200, 200);
    private static final Color EXIT_BORDER_COLOR = new Color(200, 50, 50);
    private static final Font BUTTON_FONT = new Font("Segoe UI",
Font.PLAIN, 14);
    private static final Font CELL_FONT = new Font("Segoe UI", Font.BOLD,
14);
    private static final Font LOG_FONT = new Font("Consolas", Font.PLAIN,
13);
    private static final int CELL_SIZE = 60;
    private static final int CELL_GAP = 4;
    private static final int ANIMATION_DELAY = 800;

    private State initialState;
    private JButton loadButton, solveButton, resetButton, speedButton;
    private JComboBox<String> algCombo;
    private JComboBox<String> heuristicCombo;
    private JPanel boardPanel;
    private JLabel statusLabel;
    private JTextArea logArea;
    private JScrollPane logScrollPane;

    private List<GameObject.State> pathStates;
    private int currentStep;
    private javax.swing.Timer animator;
    private int animationSpeed = ANIMATION_DELAY;
    private Map<Character, Color> pieceColors = new HashMap<>();

    public MainGUI() {
        super("Rush Hour Puzzle Solver");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout(10, 10));
        getContentPane().setBackground(BACKGROUND_COLOR);

        initComponents();
        layoutComponents();
        setupEventHandlers();

        pack();
        setMinimumSize(new Dimension(600, 700));
```

```
        setLocationRelativeTo(null);
        setVisible(true);
    }

    private void initComponents() {
        loadButton = createStyledButton("Load Puzzle", new Color(100, 150,
230));
        solveButton = createStyledButton("Solve", new Color(100, 200,
130));
        resetButton = createStyledButton("Reset", new Color(230, 150,
100));
        speedButton = createStyledButton("Speed: Normal", new Color(150,
150, 230));
        solveButton.setEnabled(false);
        resetButton.setEnabled(false);

        String[] algs = { "UCS", "GBFS", "A* Search" };
        algCombo = new JComboBox<>(algs);
        algCombo.setSelectedIndex(0);

        String[] heuristics = { "Exit distance only", "Blockers count
only", "Both heuristics" };
        heuristicCombo = new JComboBox<>(heuristics);
        heuristicCombo.setEnabled(false);
        algCombo.addActionListener(e -> {
            String algo = (String) algCombo.getSelectedItem();
            boolean needsHeuristic = algo.equals("GBFS") || algo.equals("A*
Search");
            heuristicCombo.setEnabled(needsHeuristic);
        });

        statusLabel = new JLabel("Load a puzzle file to begin");
        statusLabel.setFont(new Font("Segoe UI", Font.ITALIC, 12));
        statusLabel.setForeground(new Color(100, 100, 100));

        boardPanel = new JPanel(null);
        boardPanel.setBackground(BOARD_COLOR);
        boardPanel.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(new Color(200, 200, 200),
```

```
1),
                BorderFactory.createEmptyBorder(10, 10, 10, 10)));

        logArea = new JTextArea();
        logArea.setEditable(false);
        logArea.setFont(LOG_FONT);
        logArea.setBackground(new Color(250, 250, 250));
        logArea.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        logScrollPane = new JScrollPane(logArea);
        logScrollPane.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(new Color(200, 200, 200),
1),
                BorderFactory.createEmptyBorder(0, 0, 0, 0)));
    }

    private JButton createStyledButton(String text, Color bgColor) {
        JButton btn = new JButton(text);
        btn.setFont(BUTTON_FONT);
        btn.setBackground(bgColor);
        btn.setForeground(Color.WHITE);
        btn.setFocusPainted(false);
        btn.setBorderPainted(false);
        btn.setOpaque(true);
        btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        btn.setBorder(new EmptyBorder(8, 16, 8, 16));
        return btn;
    }

    private void layoutComponents() {
        JPanel controlPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 8,
8));
        controlPanel.setBackground(BACKGROUND_COLOR);
        controlPanel.add(new JLabel("Algorithm:"));
        controlPanel.add(algCombo);
        controlPanel.add(new JLabel("Heuristic:"));
        controlPanel.add(heuristicCombo);
        controlPanel.add(loadButton);
        controlPanel.add(solveButton);
        controlPanel.add(resetButton);
```

```
        controlPanel.add(speedButton);

        JPanel topPanel = new JPanel(new BorderLayout());
        topPanel.setBackground(BACKGROUND_COLOR);
        topPanel.add(controlPanel, BorderLayout.CENTER);
        topPanel.add(statusLabel, BorderLayout.SOUTH);
        topPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

        add(topPanel, BorderLayout.NORTH);
        add(boardPanel, BorderLayout.CENTER);
        add(logScrollPane, BorderLayout.SOUTH);

        boardPanel.setPreferredSize(new Dimension(400, 400));
        logScrollPane.setPreferredSize(new Dimension(400, 150));
    }

    private void setupEventHandlers() {
        loadButton.addActionListener(e -> onLoad());
        solveButton.addActionListener(e -> onSolve());
        resetButton.addActionListener(e -> resetAnimation());
        speedButton.addActionListener(e -> toggleSpeed());
    }

    private void toggleSpeed() {
        if (animationSpeed == ANIMATION_DELAY) {
            animationSpeed = ANIMATION_DELAY / 2;
            speedButton.setText("Speed: Fast");
        } else if (animationSpeed == ANIMATION_DELAY / 2) {
            animationSpeed = ANIMATION_DELAY / 4;
            speedButton.setText("Speed: Very Fast");
        } else {
            animationSpeed = ANIMATION_DELAY;
            speedButton.setText("Speed: Normal");
        }
        if (animator != null && animator.isRunning()) {
            animator.setDelay(animationSpeed);
        }
    }
```

```
    private void resetAnimation() {
        if (animator != null && animator.isRunning()) {
            animator.stop();
        }
        if (pathStates != null && !pathStates.isEmpty()) {
            currentStep = 0;
            renderBoard(pathStates.get(0).buildBoard(), '\0');
            logArea.append("Animation reset to initial state.\n");
            solveButton.setEnabled(true);
            updateStatus("Ready to solve or load a new puzzle");
        }
    }

    private void onLoad() {
        JFileChooser chooser = new JFileChooser("test");
        chooser.setFileFilter(new FileNameExtensionFilter("Text Files",
"txt"));
        if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            try {
                String path = chooser.getSelectedFile().getAbsolutePath();
                String name = chooser.getSelectedFile().getName();
                initialState = Parser.loadState(path);
                assignPieceColors(initialState);
                renderBoard(initialState.buildBoard(), '\0');
                logArea.setText("Loaded " + name + "\n");
                solveButton.setEnabled(true);
                resetButton.setEnabled(false);
                updateStatus("Puzzle loaded successfully. Click 'Solve'.");
            } catch (IOException | IllegalArgumentException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
                updateStatus("Error loading puzzle");
            }
        }
    }

    private void onSolve() {
        solveButton.setEnabled(false);
        resetButton.setEnabled(true);
```

```
        updateStatus("Solving puzzle...");

        new SwingWorker<Void, String>() {
            private List<GameObject.State> solution;
            private boolean found;
            private int nodesExplored;
            private double runtime;

            @Override
            protected Void doInBackground() {
                String algo = (String) algCombo.getSelectedItem();
                publish("Solving with " + algo + "...");
                if ("UCS".equals(algo)) {
                    UCS solver = new UCS();
                    found = solver.solve(initialState);
                    nodesExplored = solver.nodesExplored;
                    runtime = solver.getRuntime();
                    solution = buildPath(solver.finalState);
                } else if ("GBFS".equals(algo)) {
                    int choice = heuristicCombo.getSelectedIndex();
                    GBFS solver = new GBFS(choice);
                    found = solver.solve(initialState);
                    nodesExplored = solver.nodesExplored;
                    runtime = solver.getRuntime();
                    solution = buildPath(solver.finalState);
                }
                else if ("A* Search".equals(algo)) {
                    int choice = heuristicCombo.getSelectedIndex();
                    AStar solver = new AStar(choice);
                    found = solver.solve(initialState);
                    nodesExplored = solver.nodesExplored;
                    runtime = solver.getRuntime();
                    solution = buildPath(solver.finalState);
                }
                return null;
            }

            @Override
            protected void process(List<String> chunks) {
```

```
                chunks.forEach(line -> logArea.append(line + "\n"));
            }

            @Override
            protected void done() {
                logArea.append("Found: " + found + " | Nodes: " +
nodesExplored + "\n");
                logArea.append(String.format("Runtime: %.3f ms\n\n",
runtime));
                if (found && solution != null) {
                    pathStates = solution;
                    startAnimation();
                } else {
                    logArea.append("No solution found.\n");
                    updateStatus("No solution");
                    solveButton.setEnabled(true);
                }
            }
        }.execute();
    }

    private List<GameObject.State> buildPath(GameObject.State end) {
        LinkedList<GameObject.State> list = new LinkedList<>();
        GameObject.State cur = end;
        while (cur != null) {
            list.addFirst(cur);
            cur = cur.parent;
        }
        return list;
    }

    private void startAnimation() {
        currentStep = 0;
        animator = new javax.swing.Timer(animationSpeed, new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (currentStep < pathStates.size()) {
                    State s = pathStates.get(currentStep);
```

```
                    renderBoard(s.buildBoard(), s.carId);
                    logArea.append(describeMove(s, currentStep) + "\n");
                    updateStatus("Step " + currentStep + " of " +
(pathStates.size() - 1));
                    currentStep++;

logArea.setCaretPosition(logArea.getDocument().getLength());
                } else {
                    animator.stop();
                    logArea.append("Solution complete! " +
(pathStates.size() - 1) + " moves total.\n");
                    updateStatus("Complete");
                    solveButton.setEnabled(true);
                }
            }
        });
        animator.setInitialDelay(500);
        animator.start();
    }

    private String describeMove(State s, int step) {
        if (step == 0)
            return "Initial state";
        StringBuilder sb = new StringBuilder();
        sb.append("Step ").append(step).append(": Move car
").append(s.carId);
        switch (s.direction) {
            case 0:
                sb.append(" ← left");
                break;
            case 1:
                sb.append(" → right");
                break;
            case 2:
                sb.append(" ↑ up");
                break;
            case 3:
                sb.append(" ↓ down");
                break;
```

```
        }
        return sb.toString();
    }

    private void renderBoard(char[][] board, char highlightId) {
        boardPanel.removeAll();
        int h = board.length, w = board[0].length;
        Car mainCar = initialState.cars.get('P');
        boolean horiz = mainCar.isHorizontal;
        int exitRow = State.exitRow, exitCol = State.exitCol;

        int newCols = horiz ? w + 1 : w;
        int newRows = horiz ? h : h + 1;
        int shiftRight = horiz && exitCol == 0 ? 1 : 0;
        int shiftDown = !horiz && exitRow == 0 ? 1 : 0;

        boardPanel.setPreferredSize(new Dimension(
                newCols * (CELL_SIZE + CELL_GAP) + CELL_GAP + 20,
                newRows * (CELL_SIZE + CELL_GAP) + CELL_GAP + 20));

        for (int i = 0; i < h; i++) {
            for (int j = 0; j < w; j++) {
                int x = (j + shiftRight) * (CELL_SIZE + CELL_GAP) +
CELL_GAP;
                int y = (i + shiftDown) * (CELL_SIZE + CELL_GAP) +
CELL_GAP;
                JPanel cell = createBoardCell(board[i][j], highlightId);
                cell.setBounds(x, y, CELL_SIZE, CELL_SIZE);
                boardPanel.add(cell);
            }
        }
        int exitX, exitY;
        String arrow;
        if (horiz) {
            exitX = (exitCol + (exitCol == 0 ? 0 : 1)) * (CELL_SIZE +
CELL_GAP) + CELL_GAP;
            exitY = exitRow * (CELL_SIZE + CELL_GAP) + CELL_GAP;
            arrow = exitCol == 0 ? "←" : "→";
        } else {
```

```
            exitX = exitCol * (CELL_SIZE + CELL_GAP) + CELL_GAP;
            exitY = (exitRow + (exitRow == 0 ? 0 : 1)) * (CELL_SIZE +
CELL_GAP) + CELL_GAP;
            arrow = exitRow == 0 ? "↑" : "↓";
        }
        JPanel exitCell = new JPanel(new BorderLayout());
        exitCell.setBackground(EXIT_COLOR);

exitCell.setBorder(BorderFactory.createLineBorder(EXIT_BORDER_COLOR, 3));
        exitCell.setBounds(exitX, exitY, CELL_SIZE, CELL_SIZE);
        JLabel lbl = new JLabel("EXIT " + arrow, SwingConstants.CENTER);
        lbl.setFont(new Font("Segoe UI", Font.BOLD, 16));
        lbl.setForeground(EXIT_BORDER_COLOR);
        exitCell.add(lbl, BorderLayout.CENTER);
        boardPanel.add(exitCell);

        boardPanel.revalidate();
        boardPanel.repaint();
        pack();
    }

    private JPanel createBoardCell(char cellChar, char highlightId) {
        JPanel cell = new JPanel(new BorderLayout());
        cell.setOpaque(true);
        if (cellChar == '.') {
            cell.setBackground(BOARD_COLOR);
            cell.setBorder(BorderFactory.createLineBorder(new Color(200,
200, 200), 1));
        } else {
            Color carColor = pieceColors.getOrDefault(cellChar,
Color.LIGHT_GRAY);
            cell.setBackground(carColor);
            if (cellChar == highlightId) {
                cell.setBorder(BorderFactory.createCompoundBorder(
                        BorderFactory.createLineBorder(new Color(50, 50,
50), 3),
                        BorderFactory.createEmptyBorder(2, 2, 2, 2)));
            } else {
                cell.setBorder(BorderFactory.createCompoundBorder(
```

```
                        BorderFactory.createLineBorder(carColor.darker(),
2),
                        BorderFactory.createEmptyBorder(2, 2, 2, 2)));
            }
            JLabel carLabel = new JLabel(String.valueOf(cellChar));
            carLabel.setFont(CELL_FONT);
            carLabel.setForeground(getLabelColor(carColor));
            carLabel.setHorizontalAlignment(SwingConstants.CENTER);
            cell.add(carLabel, BorderLayout.CENTER);
            if (cellChar == 'P') {
                JLabel target = new JLabel("♦");
                target.setFont(new Font("Segoe UI", Font.BOLD, 9));
                target.setForeground(getLabelColor(carColor));
                target.setHorizontalAlignment(SwingConstants.RIGHT);
                target.setVerticalAlignment(SwingConstants.TOP);
                cell.add(target, BorderLayout.NORTH);
            }
        }
        return cell;
    }

    private Color getLabelColor(Color bg) {
        double lum = (0.299 * bg.getRed() + 0.587 * bg.getGreen() + 0.114 *
bg.getBlue()) / 255;
        return lum > 0.6 ? Color.BLACK : Color.WHITE;
    }

    private void assignPieceColors(State state) {
        pieceColors.clear();
        RandomColorGenerator gen = new RandomColorGenerator();
        for (Character id : state.cars.keySet()) {
            pieceColors.put(id, gen.nextColor());
        }
    }

    private void updateStatus(String msg) {
        statusLabel.setText(msg);
    }
```

```
    public static void main(String[] args) {
        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {
        }
        SwingUtilities.invokeLater(MainGUI::new);
    }
}

class RandomColorGenerator {
    private float hue = 0, sat = 0.8f, bri = 0.95f;

    public Color nextColor() {
        Color c = Color.getHSBColor(hue, sat, bri);
        hue = (hue + 0.618033988749895f) % 1.0f;
        return c;
    }
}
```

Nama	Jenis	Deskripsi	Return
MainGUI	Class	GUI utama aplikasi yang menampilkan kontrol pemilihan algoritma, papan puzzle, dan log.	—
MainGUI()	Constructor	Inisialisasi GUI, layout, event handler, dan membuka jendela utama.	—
initComponents()	Method (private)	Membuat dan menginisialisasi semua komponen GUI seperti tombol, combo box, label, dan panel.	void
createStyledButton(...)	Method (private)	Membuat tombol dengan gaya dan warna tertentu.	JButton

layoutComponents()	Method (private)	Menyusun komponen GUI ke dalam frame.	void
setupEventHandlers()	Method (private)	Menyambungkan aksi tombol ke handler seperti load, solve, reset, dan speed toggle.	void
toggleSpeed()	Method (private)	Mengubah kecepatan animasi di antara normal, fast, dan very fast.	void
resetAnimation()	Method (private)	Menghentikan animasi dan mengembalikan ke langkah awal.	void
onLoad()	Method (private)	Menangani aksi pemilihan file, parsing puzzle, dan render awal papan.	void
onSolve()	Method (private)	Menjalankan algoritma pencarian dalam thread terpisah menggunakan SwingWorker.	void
buildPath(State end)	Method (private)	Membangun daftar path dari state akhir ke awal (solusi).	List<State>
startAnimation()	Method (private)	Memulai animasi visual perpindahan langkah-langkah solusi.	void
describeMove(State, int)	Method (private)	Mengembalikan string deskriptif untuk langkah tertentu dalam solusi.	String
renderBoard(char[][], char)	Method (private)	Merender ulang papan permainan berdasarkan state yang diberikan.	void
createBoardCell(char, char)	Method (private)	Membuat satu sel papan (panel) dengan warna dan ID mobil.	JPanel
getLabelColor(Color)	Method (private)	Menentukan warna teks berdasarkan brightness latar belakang.	Color
assignPieceColors(State)	Method (private)	Mengisi map warna acak untuk setiap mobil dalam puzzle.	void
updateStatus(String)	Method (private)	Memperbarui label status pengguna.	void

main(String[])	Method (public static)	Entry point aplikasi. Menjalankan GUI di thread Swing.	void

Komponen GUI

Nama	Tipe	Deskripsi
boardPanel	JPanel	Panel utama yang merender papan puzzle
logArea	JTextArea	Area untuk mencetak log langkah, hasil pencarian
algCombo	JComboBox<String>	Pilihan algoritma: UCS, GBFS, A*
heuristicCombo	JComboBox<String>	Pilihan heuristik untuk GBFS dan A*
solveButton	JButton	Tombol untuk menjalankan algoritma
loadButton	JButton	Tombol untuk load puzzle dari file
resetButton	JButton	Tombol untuk reset animasi
speedButton	JButton	Tombol untuk mengganti kecepatan animasi
statusLabel	JLabel	Label status informasi saat ini
pathStates	List<State>	Daftar state dari solusi
animator	Timer	Timer animasi langkah-langkah

Inner Class RandomColorGenerator

Nama	Jenis	Deskripsi	Return
RandomColorGenerator	Class	Kelas pembantu untuk menghasilkan warna-warna HSB unik secara siklik.	—
float hue, sat, bri	Field	Properti HSB yang akan terus berubah untuk menghindari warna yang sama.	—

nextColor()	Method	Menghasilkan warna baru dengan perubahan kecil pada nilai hue.	Color

Util Package

3.2.8. ColorUtil.java

```java
package util;
public class ColorUtil {
    public static final String RESET = "\u001B[0m";
    public static final String RED = "\u001B[31m";
    public static final String GREEN = "\u001B[32m";
    public static final String YELLOW = "\u001B[33m";
    public static final String BLUE = "\u001B[34m";

    public static String colorize(char text, String colorCode) {
        return colorCode + text + RESET;
    }
}
```

Nama	Jenis	Deskripsi	Return
ColorUtil	Class	Kelas utilitas untuk memberi warna ANSI pada teks karakter di console (khusus terminal ANSI).	—
RESET	Constant (public)	Escape code untuk reset warna ke default.	String
RED	Constant (public)	Escape code warna merah ANSI.	String
GREEN	Constant (public)	Escape code warna hijau ANSI.	String

YELLOW	Constant (public)	Escape code warna kuning ANSI.	String
BLUE	Constant (public)	Escape code warna biru ANSI.	String
colorize(char text, String colorCode)	Method (public static)	Memberikan warna tertentu ke karakter text menggunakan escape ANSI, lalu mereset.	String

3.2.8. Main.java

```
import java.io.IOException;
import GameObject.*;
import FileReader.*;
import Algorithm.*;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);

        // Load input state from file
        System.out.print("Enter the file name to load the state: ");
        String filePath = scan.nextLine();
        State loadedState = null;
        try {
            loadedState = Parser.loadState(filePath);
            loadedState.printBoard();
        } catch (IOException e) {
            System.out.println("Failed to read file: " + e.getMessage());
            scan.close();
            return;
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid file format: " + e.getMessage());
            scan.close();
            return;
        }

        for (Car car : loadedState.cars.values()) {
```

```
            System.out.println("Car ID: " + car.id + ", Row: " + car.row +
", Col: " + car.col);
        }
        System.out.println("Exit Row: " + State.exitRow);
        System.out.println("Exit Col: " + State.exitCol);

        // Ask user to select search algorithm
        System.out.println("\nSelect search algorithm:");
        System.out.println("1. UCS");
        System.out.println("2. GBFS");
        System.out.println("3. AStar");
        System.out.print("Enter your choice (1, 2, or 3): ");
        int algChoice = scan.nextInt();
        scan.nextLine(); // Clear the buffer

        boolean solved = false;
        if (algChoice == 1) {
            UCS solution = new UCS();
            solved = solution.solve(loadedState);
            if (solved) {
                System.out.println("Solution found!");
                System.out.println("Nodes explored: " +
solution.nodesExplored);
                System.out.printf("Runtime: %.3f ms%n",
solution.getRuntime());
            } else {
                System.out.println("No solution found.");
            }
        } else if (algChoice == 2) {
            // Ask for heuristic selection if using GBFS
            System.out.println("\nSelect heuristic option for GBFS:");
            System.out.println("0: Exit distance heuristic only");
            System.out.println("1: Blockers count heuristic only");
            System.out.println("2: Both heuristics");
            System.out.print("Enter your heuristic choice (0, 1, or 2): ");
            int heuristicChoice = scan.nextInt();
            scan.nextLine(); // Clear the buffer

            GBFS solution = new GBFS(heuristicChoice);
```

```
            solved = solution.solve(loadedState);
            if (solved) {
                System.out.println("Solution found!");
                System.out.println("Nodes explored: " +
solution.nodesExplored);
                System.out.printf("Runtime: %.3f ms%n",
solution.getRuntime());
            } else {
                System.out.println("No solution found.");
            }
        } else if (algChoice == 3) {
            // Ask for heuristic selection if using AStar
            System.out.println("\nSelect heuristic option for AStar:");
            System.out.println("0: Exit distance heuristic only");
            System.out.println("1: Blockers count heuristic only");
            System.out.println("2: Both heuristics");
            System.out.print("Enter your heuristic choice (0, 1, or 2): ");
            int heuristicChoice = scan.nextInt();
            scan.nextLine(); // Clear the buffer

            AStar solution = new AStar(heuristicChoice);
            solved = solution.solve(loadedState);
            if (solved) {
                System.out.println("Solution found!");
                System.out.println("Nodes explored: " +
solution.nodesExplored);
                System.out.printf("Runtime: %.3f ms%n",
solution.getRuntime());
            } else {
                System.out.println("No solution found.");
            }
        } else {
            System.out.println("Invalid selection. Exiting.");
        }

        scan.close();
    }
}
```

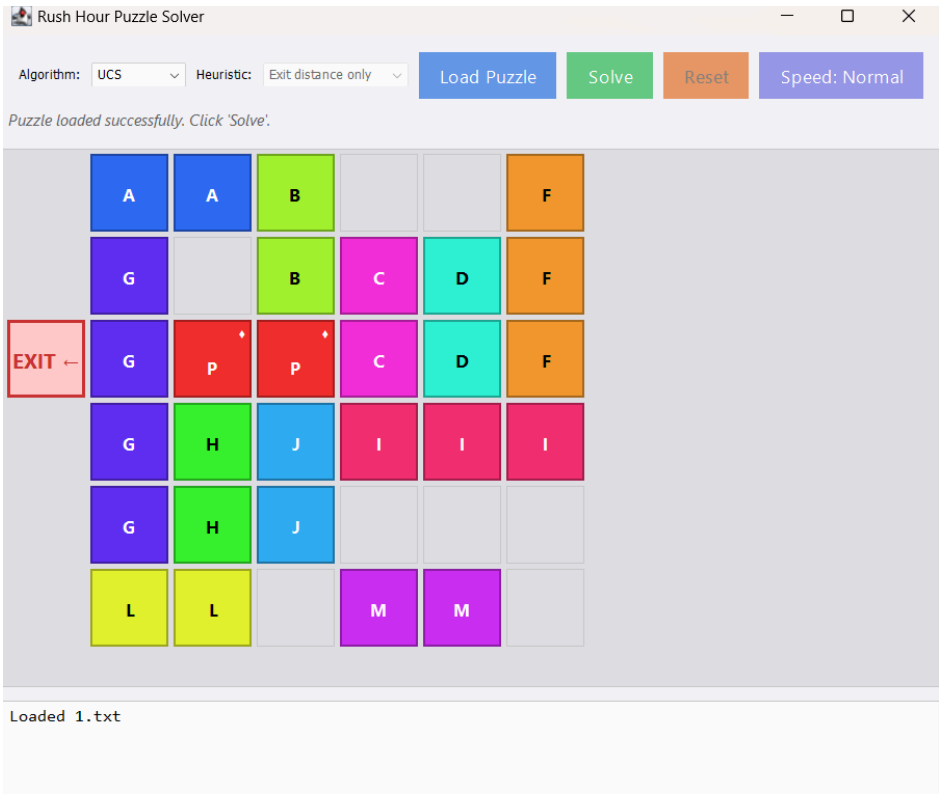
Nama	Jenis	Deskripsi	Return
Main	Class	Entry point CLI untuk menjalankan solver puzzle Rush Hour dari file dan memilih algoritma dari terminal.	—
main(String[] args)	Method (static)	Fungsi utama: 1) Load file puzzle, 2) Tampilkan info, 3) Pilih algoritma, 4) Jalankan solver dan tampilkan hasil.	void

3.3. Pengujian

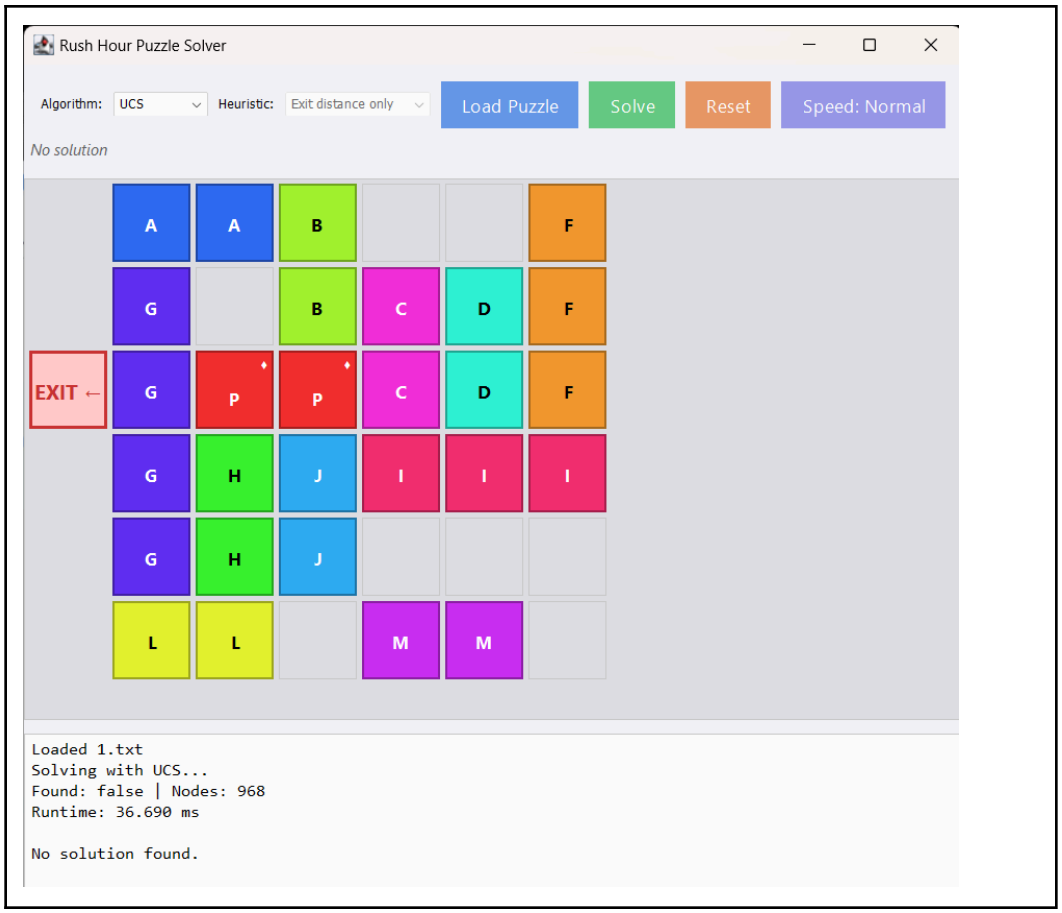
3.3.1. UCS Algorithm

Test Case 1

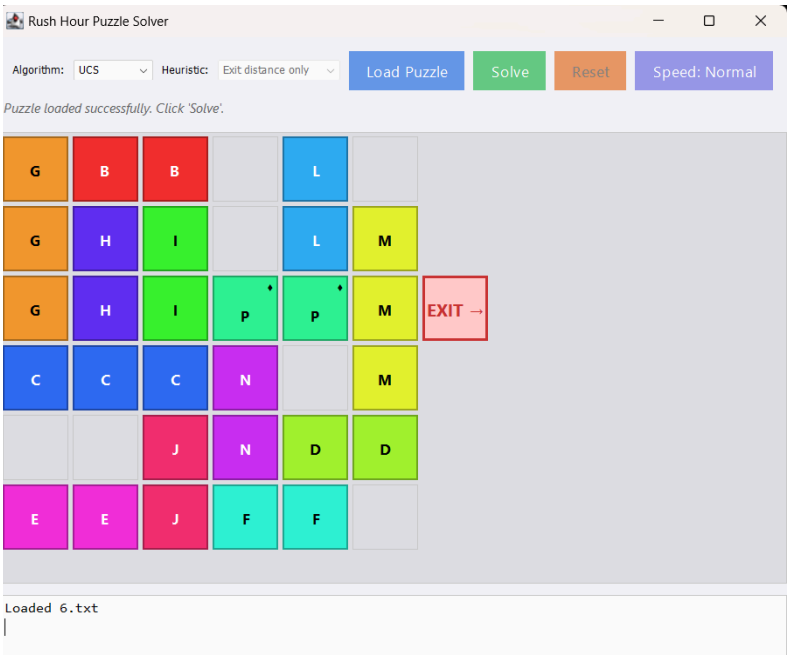
Input.txt

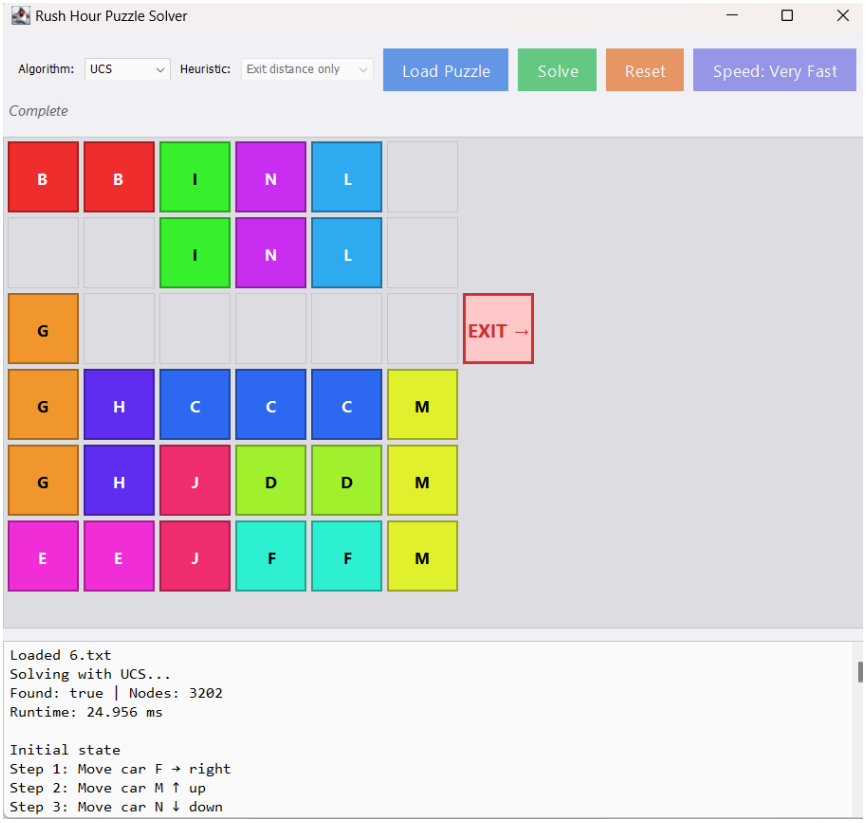

Rush Hour Puzzle Solver
Algorithm: UCS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT ←
Loaded 1.txt

Output


Rush Hour Puzzle Solver
Algorithm: UCS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
No solution
EXIT ←
Loaded 1.txt
Solving with UCS...
Found: false | Nodes: 968
Runtime: 36.690 ms
No solution found.

Test Case 2

Input.txt

Output



CLI Output

```
generating solution....
found exit
Move car F to the right
G B B . L .
G H I . L M
G H I P P M K
C C C N . M
. . J N D D
E E J . F F
-------------------------
Move car M up
G B B . L M
G H I . L M
G H I P P M K
C C C N . .
. . J N D D
E E J . F F
-------------------------
Move car N down
G B B . L M
G H I . L M
G H I P P M K
C C C . . .
. . J N D D
E E J N F F
-------------------------
Move car C to the right
G B B . L M
G H I . L M
G H I P P M K
. . . C C C
. . J N D D
E E J N F F
-------------------------
-------------------------
Move car H down
G B B . L M
G . I . L M
G . I P P M K
. H . C C C
. H J N D D
E E J N F F
-------------------------
Move car J up
G B B . L M
G . I . L M
G . I P P M K
. H J C C C
. H J N D D
E E . N F F
-------------------------
Move car E to the right
G B B . L M
G . I . L M
G . I P P M K
. H J C C C
. H J N D D
. E E N F F
-------------------------
Move car G down
. B B . L M
. . I . L M
. . I P P M K
G H J C C C
G H J N D D
G E E N F F
-------------------------
-------------------------
Move car B to the left
B B . . L M
. . I . L M
. . I P P M K
G H J C C C
G H J N D D
G E E N F F
-------------------------
Move car I up
B B I . L M
. . I . L M
. . . P P M K
G H J C C C
G H J N D D
G E E N F F
-------------------------
Move car P to the left
B B I . L M
. . I . L M
P P . . . M K
G H J C C C
G H J N D D
G E E N F F
-------------------------
Move car I down
B B . . L M
. . I . L M
P P I . . M K
G H J C C C
G H J N D D
G E E N F F
```


Move car L down
B B . . . M
. . I . L M
P P I . L M K
G H J C C C
G H J N D D
G E E N F F
Move car B to the right
. . . B B M
. . I . L M
P P I . L M K
G H J C C C
G H J N D D
G E E N F F
Move car I up
. . I B B M
. . I . L M
P P . . L M K
G H J C C C
G H J N D D
G E E N F F
Move car P to the right
. . I B B M
. . I . L M
. . P P L M K
G H J C C C
G H J N D D
G E E N F F
Move car G up
G . I B B M
G . I . L M
G . P P L M K
. H J C C C
. H J N D D
. E E N F F
Move car E to the left
G . I B B M
G . I . L M
G . P P L M K
. H J C C C
. H J N D D
E E . N F F
Move car H up
G H I B B M
G H I . L M
G . P P L M K
. . J C C C
. . J N D D
E E . N F F
Move car J down
G H I B B M
G H I . L M
G . P P L M K
. . . C C C
. . J N D D
E E J N F F
Move car C to the left
G H I B B M
G H I . L M
G . P P L M K
C C C . . .
. . J N D D
E E J N F F
Move car M down
G H I B B .
G H I . L M
G . P P L M K
C C C . . M
. . J N D D
E E J N F F
Move car B to the right
G H I . B B
G H I . L M
G . P P L M K
C C C . . M
. . J N D D
E E J N F F
Move car P to the left
G H I . B B
G H I . L M
G P P . L M K
C C C . . M
. . J N D D
E E J N F F

Move car N up
G H I N B B
G H I N L M
G P P . L M K
C C C . . M
. . J . D D
E E J . F F

Move car C to the right
G H I N B B
G H I N L M
G P P . L M K
. . C C C M
. . J . D D
E E J . F F

Move car D to the left
G H I N B B
G H I N L M
G P P . L M K
. . C C C M
. . J D D .
E E J . F F

Move car F to the left
G H I N B B
G H I N L M
G P P . L M K
. . C C C M
. . J D D .
E E J F F .

Move car P to the right
G H I N B B
G H I N L M
G . P P L M K
. . C C C M
. . J D D .
E E J F F .

Move car H down
G . I N B B
G . I N L M
G . P P L M K
. H C C C M
. H J D D .
E E J F F .

Move car P to the left
G . I N B B
G . I N L M
G P P . L M K
. H C C C M
. H J D D .
E E J F F .

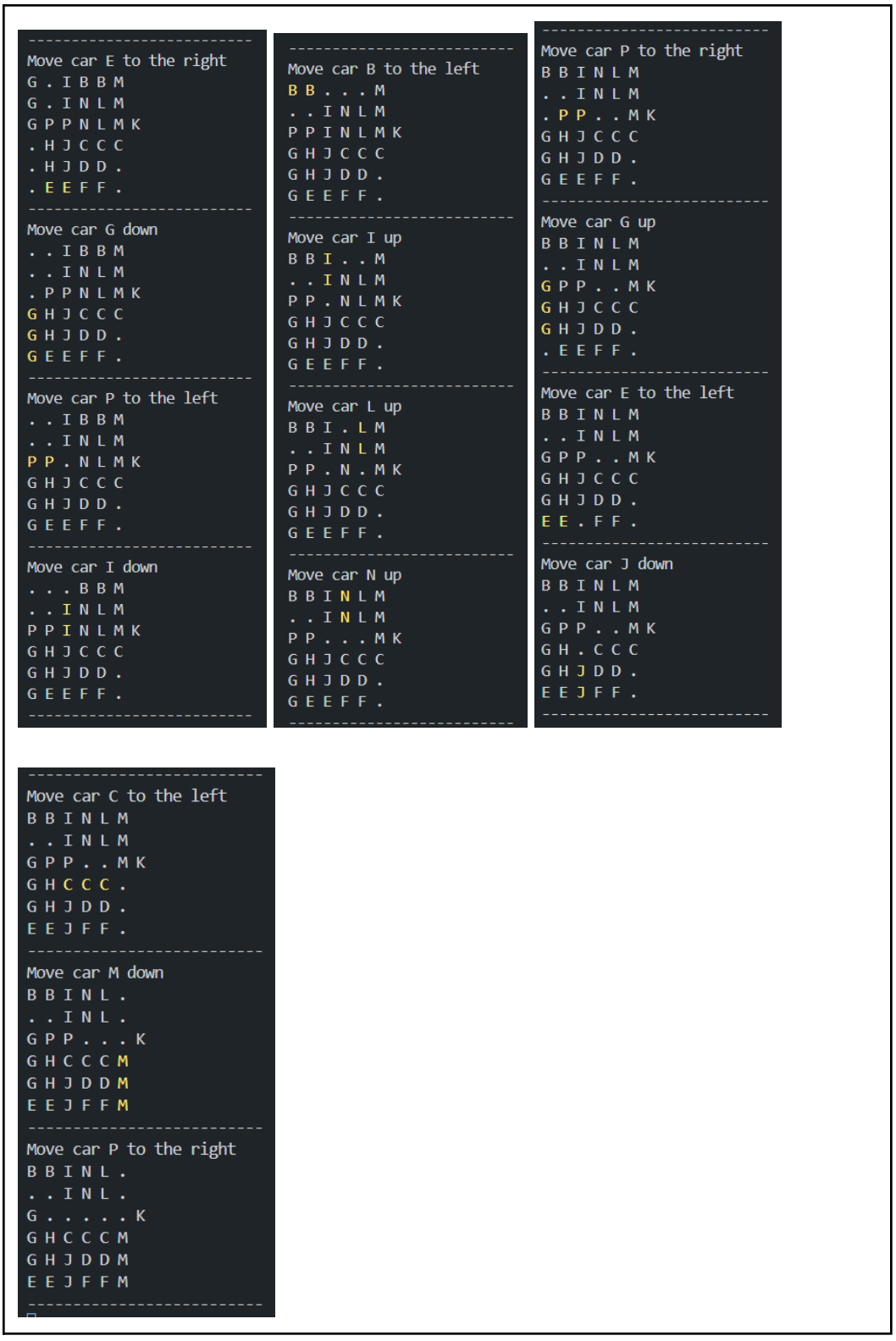
Move car N down
G . I . B B
G . I N L M
G P P N L M K
. H C C C M
. H J D D .
E E J F F .

Move car B to the left
G . I B B .
G . I N L M
G P P N L M K
. H C C C M
. H J D D .
E E J F F .

Move car M up
G . I B B M
G . I N L M
G P P N L M K
. H C C C .
. H J D D .
E E J F F .

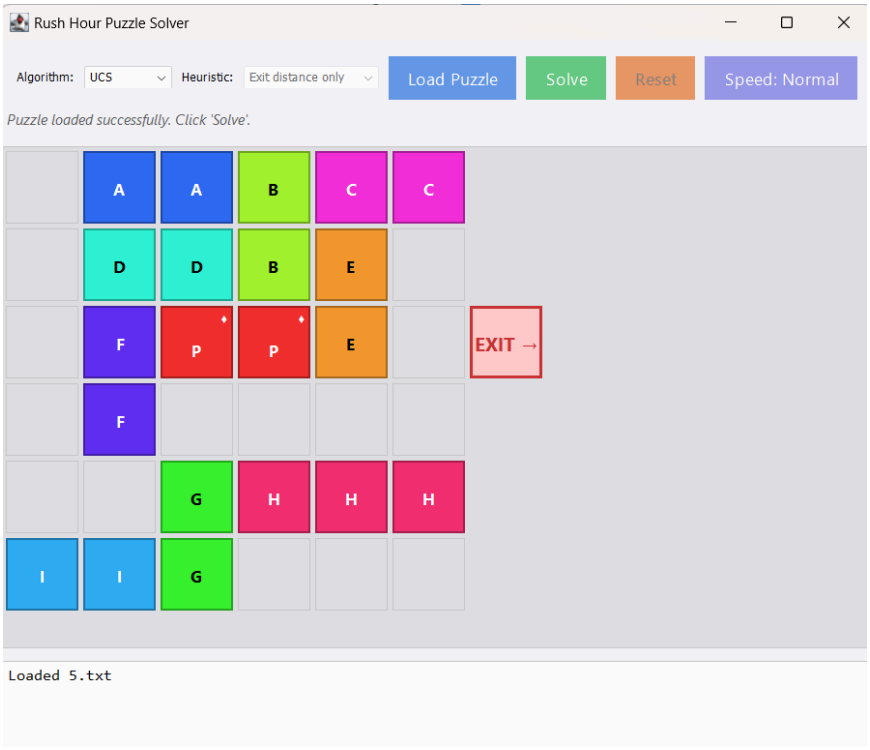
Move car C to the right
G . I B B M
G . I N L M
G P P N L M K
. H . C C C
. H J D D .
E E J F F .

Move car J up
G . I B B M
G . I N L M
G P P N L M K
. H J C C C
. H J D D .
E E . F F .

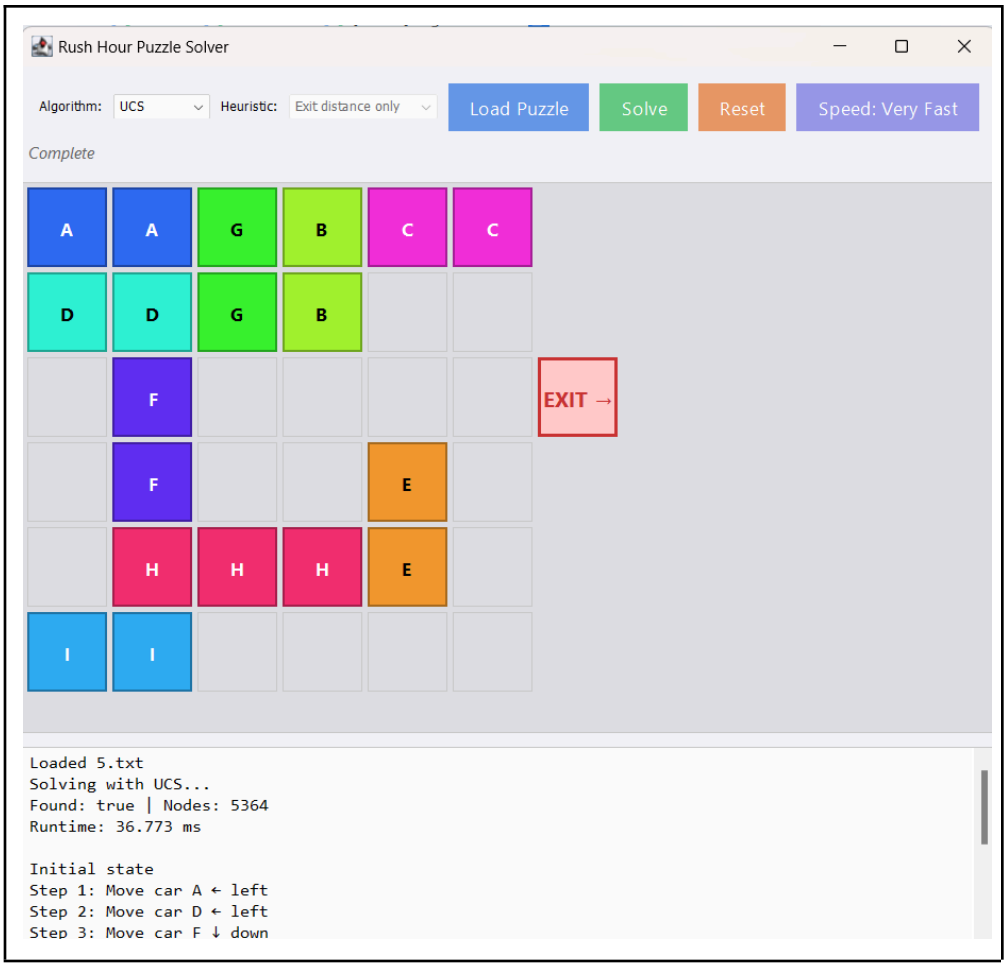

Move car E to the right
G . I B B M
G . I N L M
G P P N L M K
. H J C C C
. H J D D .
. E E F F .
Move car G down
. . I B B M
. . I N L M
. P P N L M K
G H J C C C
G H J D D .
G E E F F .
Move car P to the left
. . I B B M
. . I N L M
P P . N L M K
G H J C C C
G H J D D .
G E E F F .
Move car I down
. . . B B M
. . I N L M
P P I N L M K
G H J C C C
G H J D D .
G E E F F .
Move car B to the left
B B . . . M
. . I N L M
P P I N L M K
G H J C C C
G H J D D .
G E E F F .
Move car I up
B B I . . M
. . I N L M
P P . N L M K
G H J C C C
G H J D D .
G E E F F .
Move car L up
B B I . L M
. . I N L M
P P . N . M K
G H J C C C
G H J D D .
G E E F F .
Move car N up
B B I N L M
. . I N L M
P P . . . M K
G H J C C C
G H J D D .
G E E F F .
Move car P to the right
B B I N L M
. . I N L M
. P P . . M K
G H J C C C
G H J D D .
G E E F F .
Move car G up
B B I N L M
. . I N L M
G P P . . M K
G H J C C C
G H J D D .
. E E F F .
Move car E to the left
B B I N L M
. . I N L M
G P P . . M K
G H J C C C
G H J D D .
E E . F F .
Move car J down
B B I N L M
. . I N L M
G P P . . M K
G H . C C C
G H J D D .
E E J F F .
Move car C to the left
B B I N L M
. . I N L M
G P P . . M K
G H C C C .
G H J D D .
E E J F F .
Move car M down
B B I N L .
. . I N L .
G P P . . . K
G H C C C M
G H J D D M
E E J F F M
Move car P to the right
B B I N L .
. . I N L .
G . . . . . K
G H C C C M
G H J D D M
E E J F F M

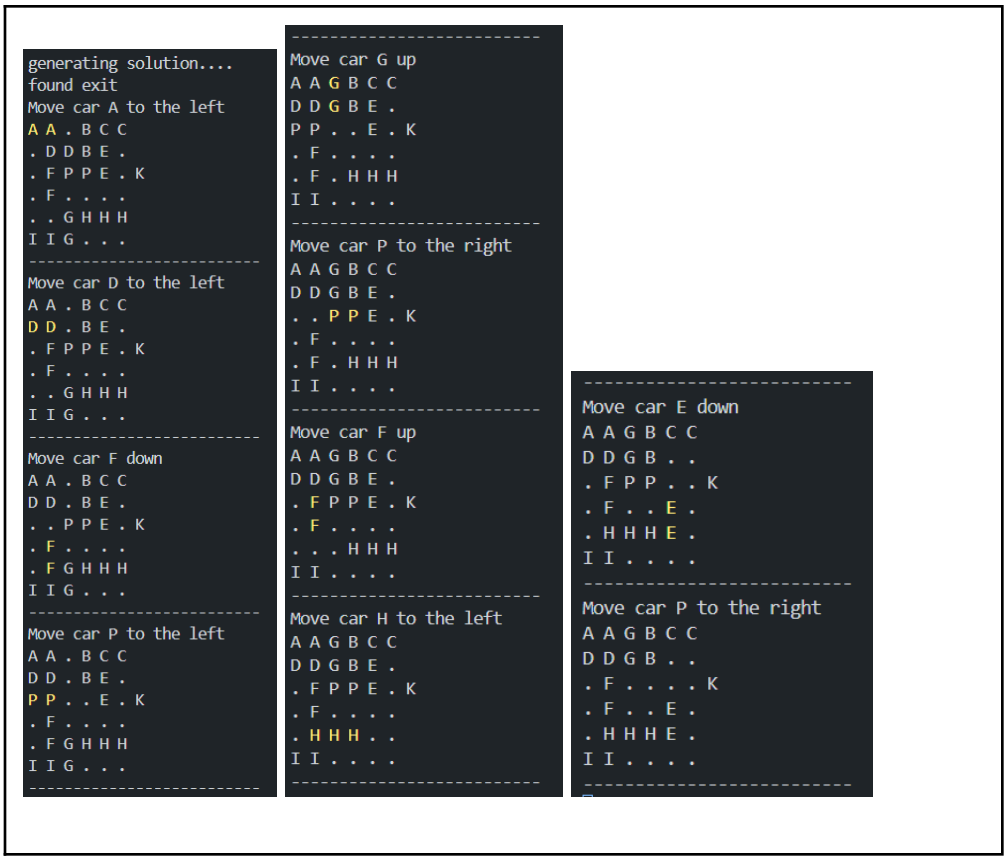
Test Case 3

Input.txt



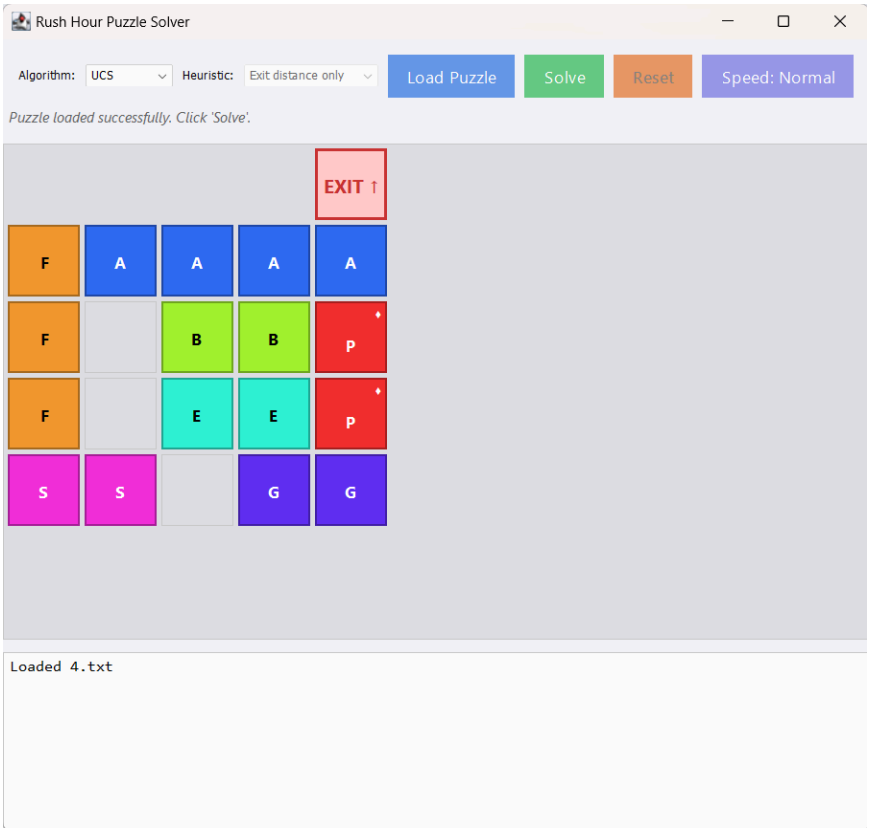
Output


Rush Hour Puzzle Solver
Algorithm: UCS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
A A G B C C
D D G B
F
EXIT →
F E
H H H E
I I
Loaded 5.txt
Solving with UCS...
Found: true | Nodes: 5364
Runtime: 36.773 ms
Initial state
Step 1: Move car A ← left
Step 2: Move car D ← left
Step 3: Move car F ↓ down



Test Case 4

Input.txt


Rush Hour Puzzle Solver
Algorithm: UCS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT ↑
F A A A A
F B B P
F E E P
S S G G
Loaded 4.txt

Output

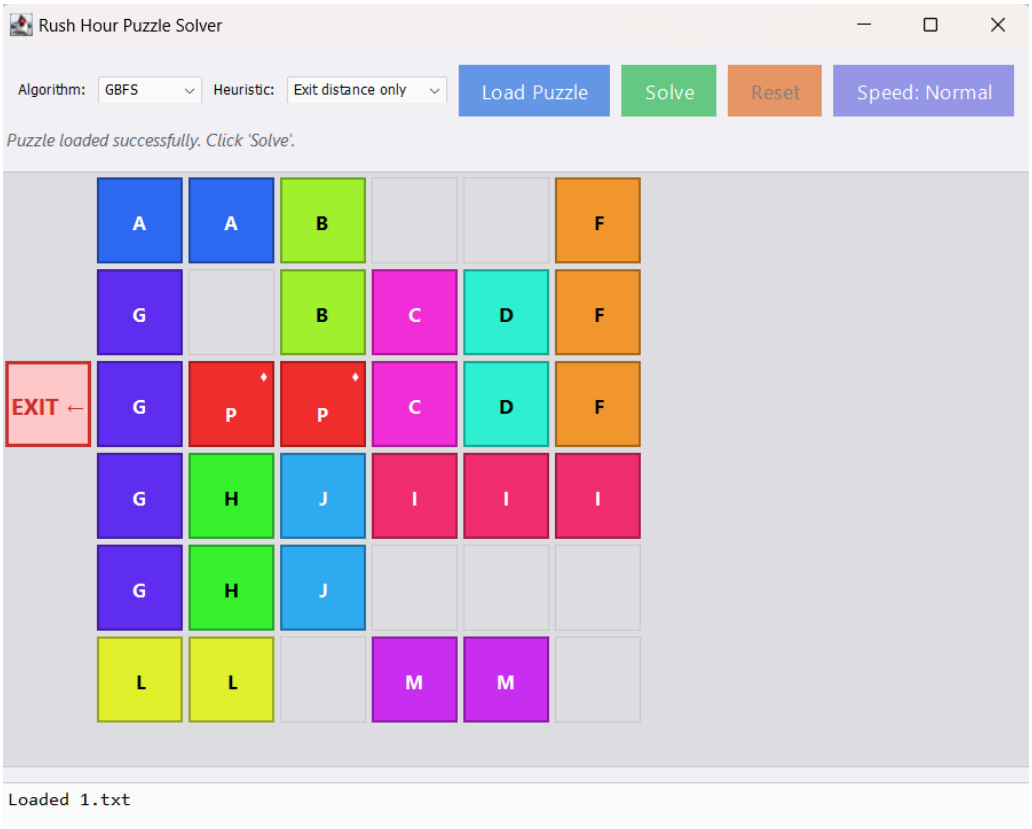
CLI OUTPUT

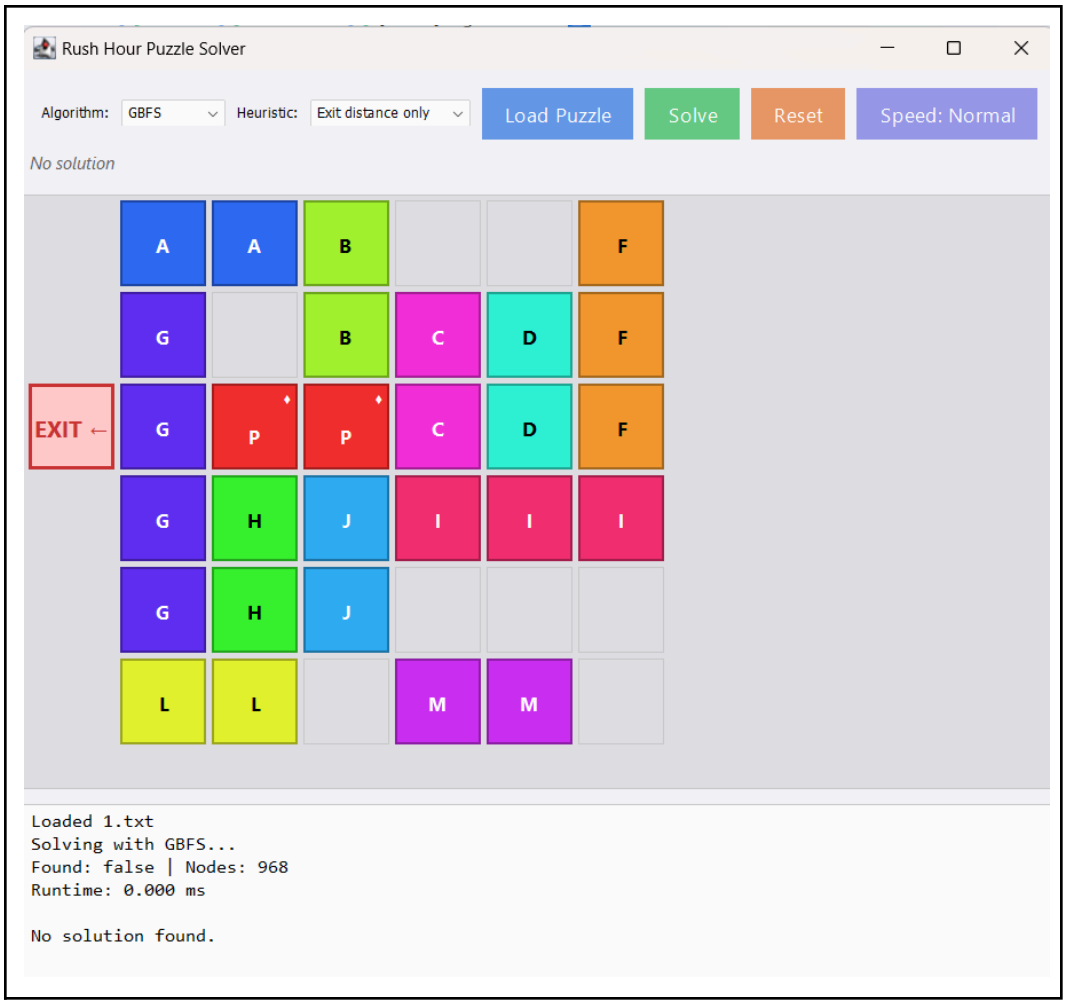
```
generating solution....
found exit
Move car S to the right
        K
F A A A A
F . B B P
F . E E P
. S S G G
-------------------------
Move car F down
        K
. A A A A
F . B B P
F . E E P
F S S G G
-------------------------
Move car A to the left
        K
A A A A .
F . B B P
F . E E P
F S S G G
-------------------------
Move car P up
        K
A A A A .
F . B B .
F . E E .
F S S G G
-------------------------
```

3.3.2. GBFS

3.3.2.1 Exit Distance Heuristic

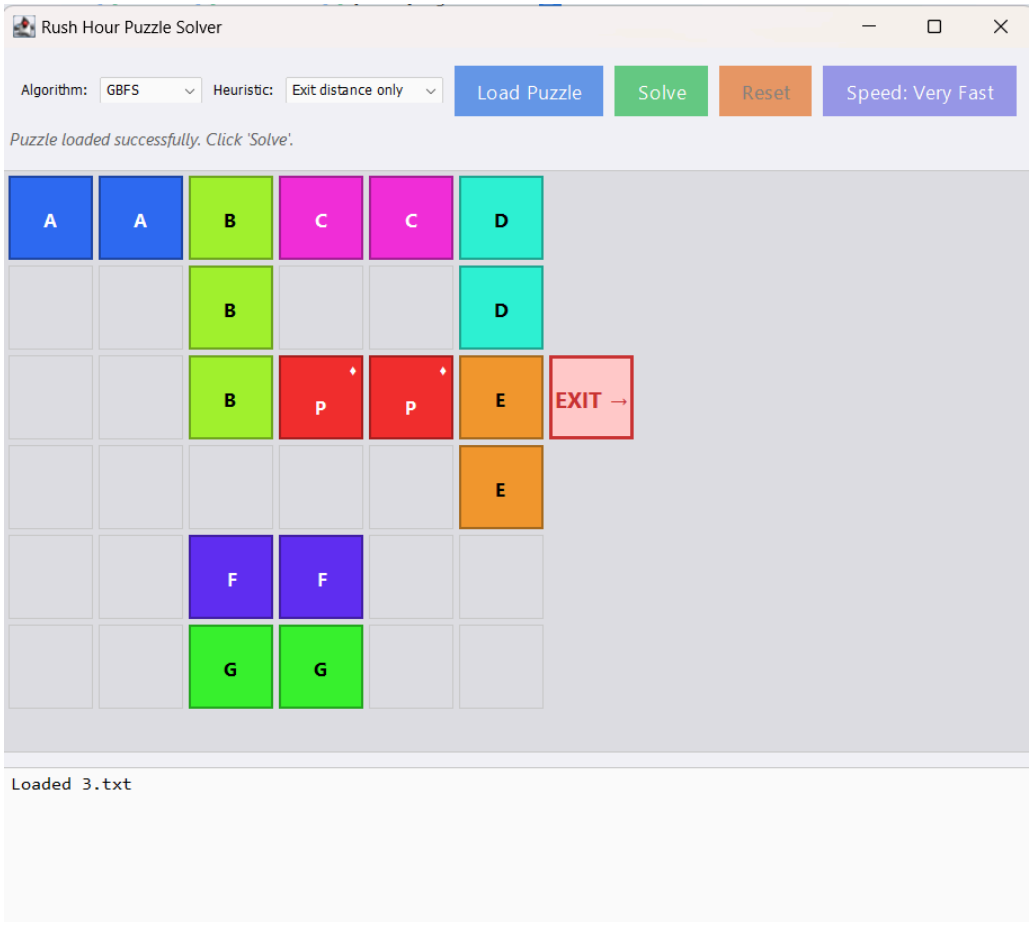
Test Case 1

Input.txt

Output

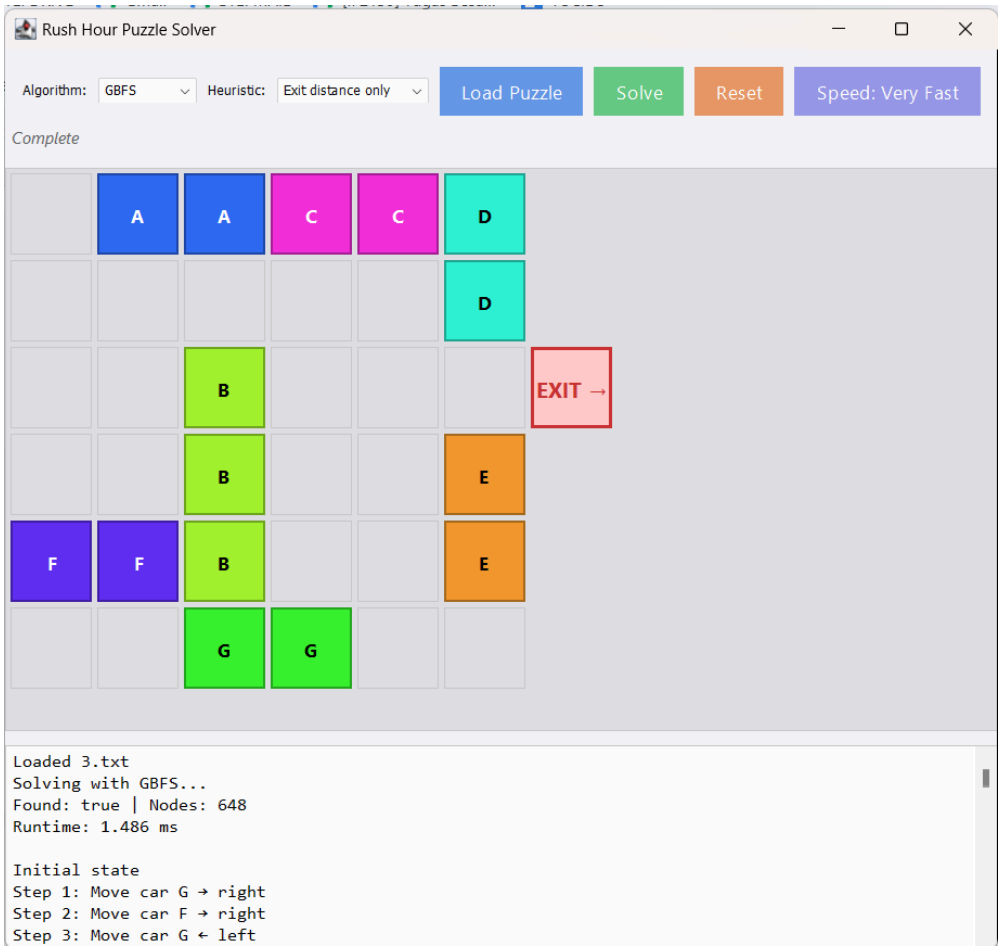


Test Case 2

Input.txt


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
A A B C C D
B D
B P P E EXIT →
E
F F
G G
Loaded 3.txt

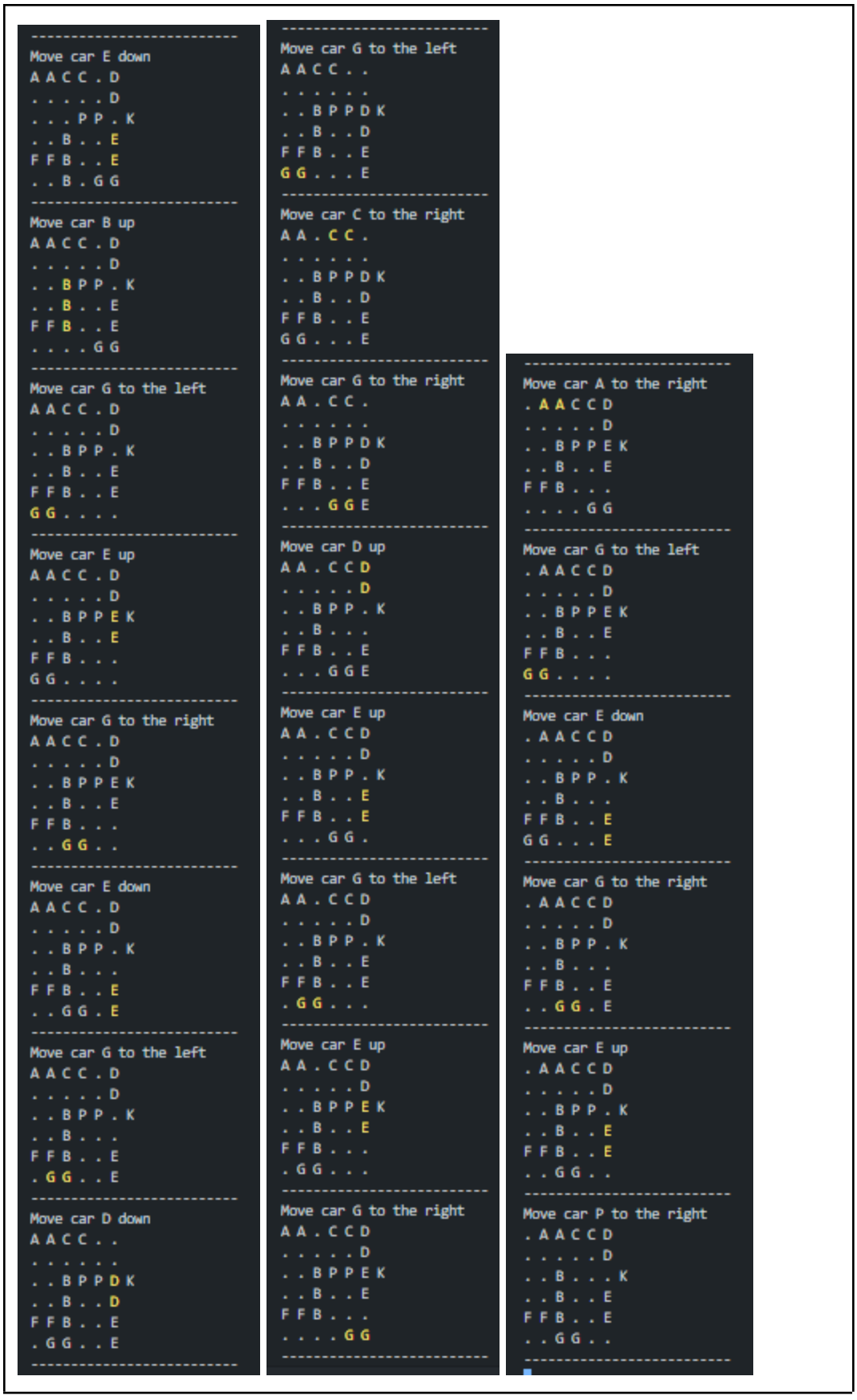
Output


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
A
A
C
C
D
D
B
EXIT →
B
E
F
F
B
E
G
G
Loaded 3.txt
Solving with GBFS...
Found: true | Nodes: 648
Runtime: 1.486 ms
Initial state
Step 1: Move car G → right
Step 2: Move car F → right
Step 3: Move car G ← left

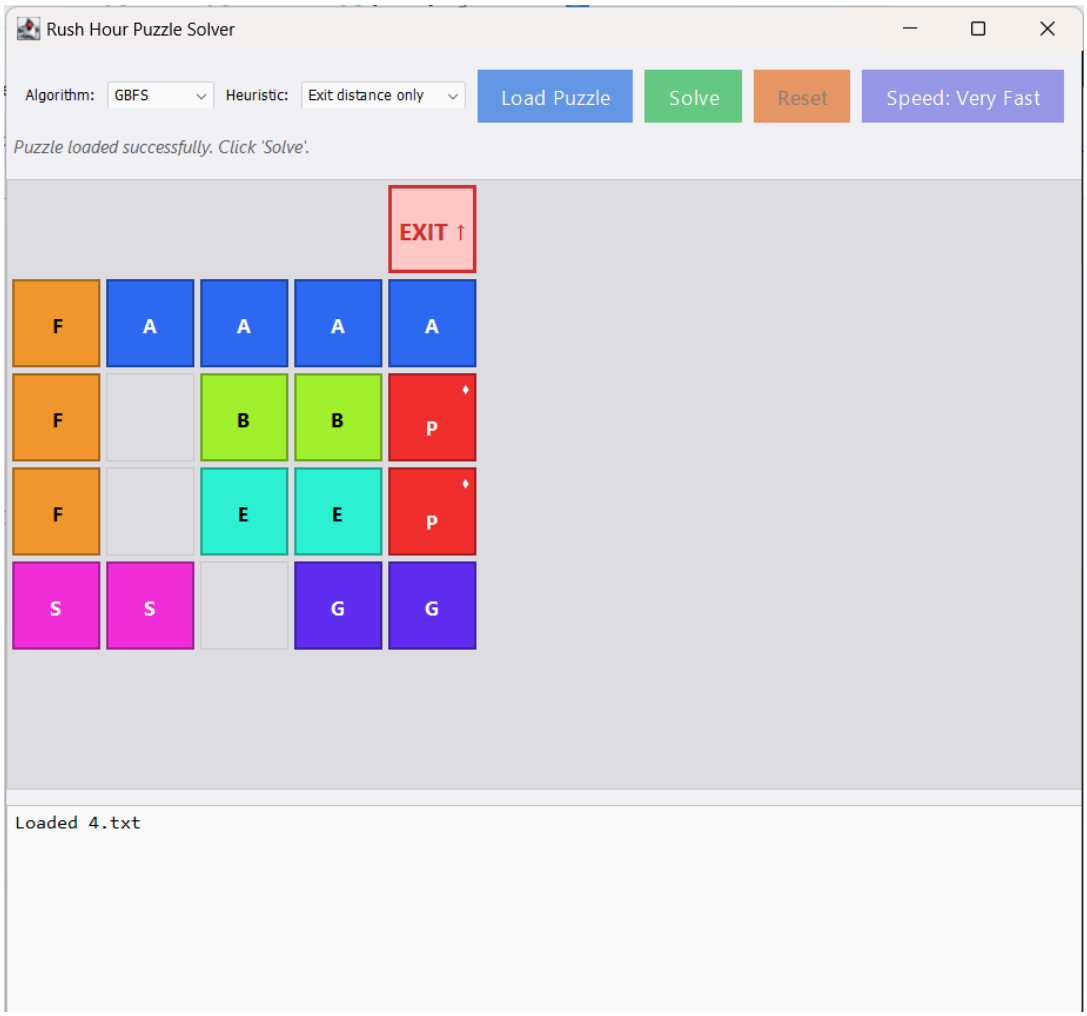
CLI OUTPUT

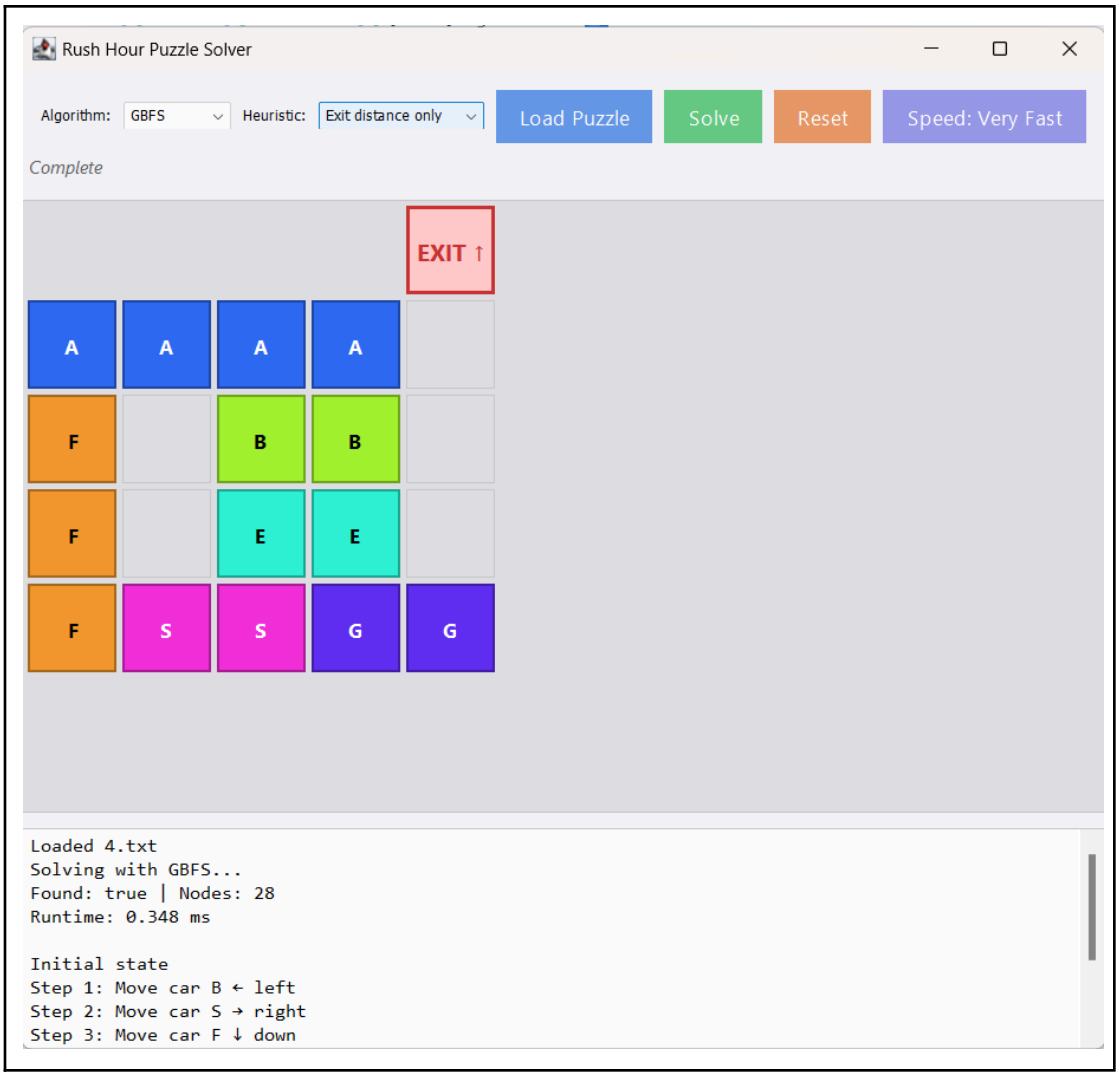

generating solution....
Exit found
Move car G to the right
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
. . F F . .
. . . . G G
Move car F to the right
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
. . . . F F
. . . . G G
Move car G to the left
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
. . . . F F
G G . . . .
Move car F to the left
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
F F . . . .
G G . . . .
Move car G to the right
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
F F . . . .
. . . G G .
Move car F to the right
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
. . . F F .
. . . G G .
Move car G to the left
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
. . . F F .
. G G . . .
Move car F to the left
A A B C C D
. . B . . D
. . B P P E K
. . . . . E
. F F . . .
. G G . . .
Move car E down
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
. F F . . E
. G G . . E
Move car G to the right
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
. F F . . E
. . . G G E
Move car F to the right
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
. . F F . E
. . . G G E
Move car G to the left
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
. . F F . E
G G . . . E
Move car F to the right
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
. . . F F E
G G . . . E
Move car G to the right
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
. . . F F E
. . G G . E
Move car F to the left
A A B C C D
. . B . . D
. . B P P . K
. . . . . .
F F . . . E
. . G G . E
Move car E up
A A B C C D
. . B . . D
. . B P P . K
. . . . . E
F F . . . E
. . G G . .
Move car G to the right
A A B C C D
. . B . . D
. . B P P . K
. . . . . E
F F . . . E
. . . . G G
Move car F to the right
A A B C C D
. . B . . D
. . B P P . K
. . . . . E
. . . F F E
. . . . G G
Move car D down
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
. . . F F E
. . . . G G
Move car G to the left
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
. . . F F E
G G . . . .
Move car F to the left
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
F F . . . E
G G . . . .
Move car G to the right
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
F F . . . E
. . . G G .

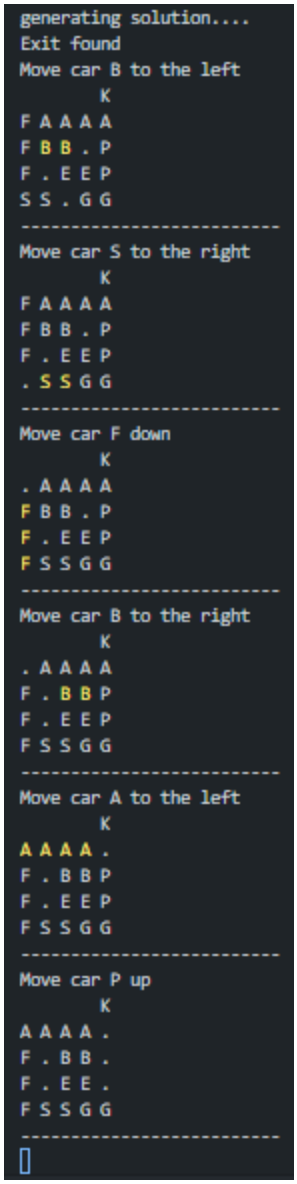

Move car F to the right
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
. . F F . E
. . . G G .
Move car G to the left
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
. . F F . E
. G G . . .
Move car F to the left
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
. F F . . E
. G G . . .
Move car G to the right
A A B C C .
. . B . . D
. . B P P D K
. . . . . E
. F F . . E
. . G G . .
Move car E down
A A B C C .
. . B . . D
. . B P P D K
. . . . . .
. F F . . E
. . G G . E
Move car G to the left
A A B C C .
. . B . . D
. . B P P D K
. . . . . .
. F F . . E
G G . . . E
Move car C to the right
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
. F F . . E
G G . . . E
Move car G to the right
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
. F F . . E
. . . G G E
Move car F to the right
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
. . . F F E
. . . G G E
Move car G to the left
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
. . . F F E
. G G . . E
Move car F to the left
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
F F . . . E
. G G . . E
Move car G to the right
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
F F . . . E
. . G G . E
Move car F to the right
A A B . C C
. . B . . D
. . B P P D K
. . . . . .
. . F F . E
. . G G . E
Move car E up
A A B . C C
. . B . . D
. . B P P D K
. . . . . E
. . F F . E
. . G G . .
Move car G to the right
A A B . C C
. . B . . D
. . B P P D K
. . . . . E
. . F F . E
. . . . G G
Move car F to the left
A A B . C C
. . B . . D
. . B P P D K
. . . . . E
F F . . . E
. . . . G G
Move car B down
A A . . C C
. . . . . D
. . . P P D K
. . B . . E
F F B . . E
. . B . G G
Move car G to the left
A A . . C C
. . . . . D
. . . P P D K
. . B . . E
F F B . . E
. . B G G .
Move car E down
A A . . C C
. . . . . D
. . . P P D K
. . B . . .
F F B . . E
. . B G G E
Move car D down
A A . . C C
. . . . . .
. . . P P D K
. . B . . D
F F B . . E
. . B G G E
Move car C to the left
A A C C . .
. . . . . .
. . . P P D K
. . B . . D
F F B . . E
. . B G G E
Move car D up
A A C C . D
. . . . . D
. . . P P . K
. . B . . .
F F B . . E
. . B G G E
Move car E up
A A C C . D
. . . . . D
. . . P P E K
. . B . . E
F F B . . .
. . B G G .
Move car G to the right
A A C C . D
. . . . . D
. . . P P E K
. . B . . E
F F B . . .
. . B . G G


Move car E down
A A C C . D
. . . . . D
. . . P P . K
. . B . . E
F F B . . E
. . B . G G
Move car B up
A A C C . D
. . . . . D
. . B P P . K
. . B . . E
F F B . . E
. . . . G G
Move car G to the left
A A C C . D
. . . . . D
. . B P P . K
. . B . . E
F F B . . E
G G . . . .
Move car E up
A A C C . D
. . . . . D
. . B P P E K
. . B . . E
F F B . . .
G G . . . .
Move car G to the right
A A C C . D
. . . . . D
. . B P P E K
. . B . . E
F F B . . .
. . G G . .
Move car E down
A A C C . D
. . . . . D
. . B P P . K
. . B . . .
F F B . . E
. . G G . E
Move car G to the left
A A C C . D
. . . . . D
. . B P P . K
. . B . . .
F F B . . E
. G G . . E
Move car D down
A A C C . .
. . . . . .
. . B P P D K
. . B . . D
F F B . . E
. G G . . E
Move car G to the left
A A C C . .
. . . . . .
. . B P P D K
. . B . . D
F F B . . E
G G . . . E
Move car C to the right
A A . C C .
. . . . . .
. . B P P D K
. . B . . D
F F B . . E
G G . . . E
Move car G to the right
A A . C C .
. . . . . .
. . B P P D K
. . B . . D
F F B . . E
. . . G G E
Move car D up
A A . C C D
. . . . . D
. . B P P . K
. . B . . .
F F B . . E
. . . G G E
Move car E up
A A . C C D
. . . . . D
. . B P P . K
. . B . . E
F F B . . E
. . . G G .
Move car G to the left
A A . C C D
. . . . . D
. . B P P . K
. . B . . E
F F B . . E
. G G . . .
Move car E up
A A . C C D
. . . . . D
. . B P P E K
. . B . . E
F F B . . .
. G G . . .
Move car G to the right
A A . C C D
. . . . . D
. . B P P E K
. . B . . E
F F B . . .
. . . . G G
Move car A to the right
. A A C C D
. . . . . D
. . B P P E K
. . B . . E
F F B . . .
. . . . G G
Move car G to the left
. A A C C D
. . . . . D
. . B P P E K
. . B . . E
F F B . . .
G G . . . .
Move car E down
. A A C C D
. . . . . D
. . B P P . K
. . B . . .
F F B . . E
G G . . . E
Move car G to the right
. A A C C D
. . . . . D
. . B P P . K
. . B . . .
F F B . . E
. . G G . E
Move car E up
. A A C C D
. . . . . D
. . B P P . K
. . B . . E
F F B . . E
. . G G . .
Move car P to the right
. A A C C D
. . . . . D
. . B . . . K
. . B . . E
F F B . . E
. . G G . .

Test Case 3

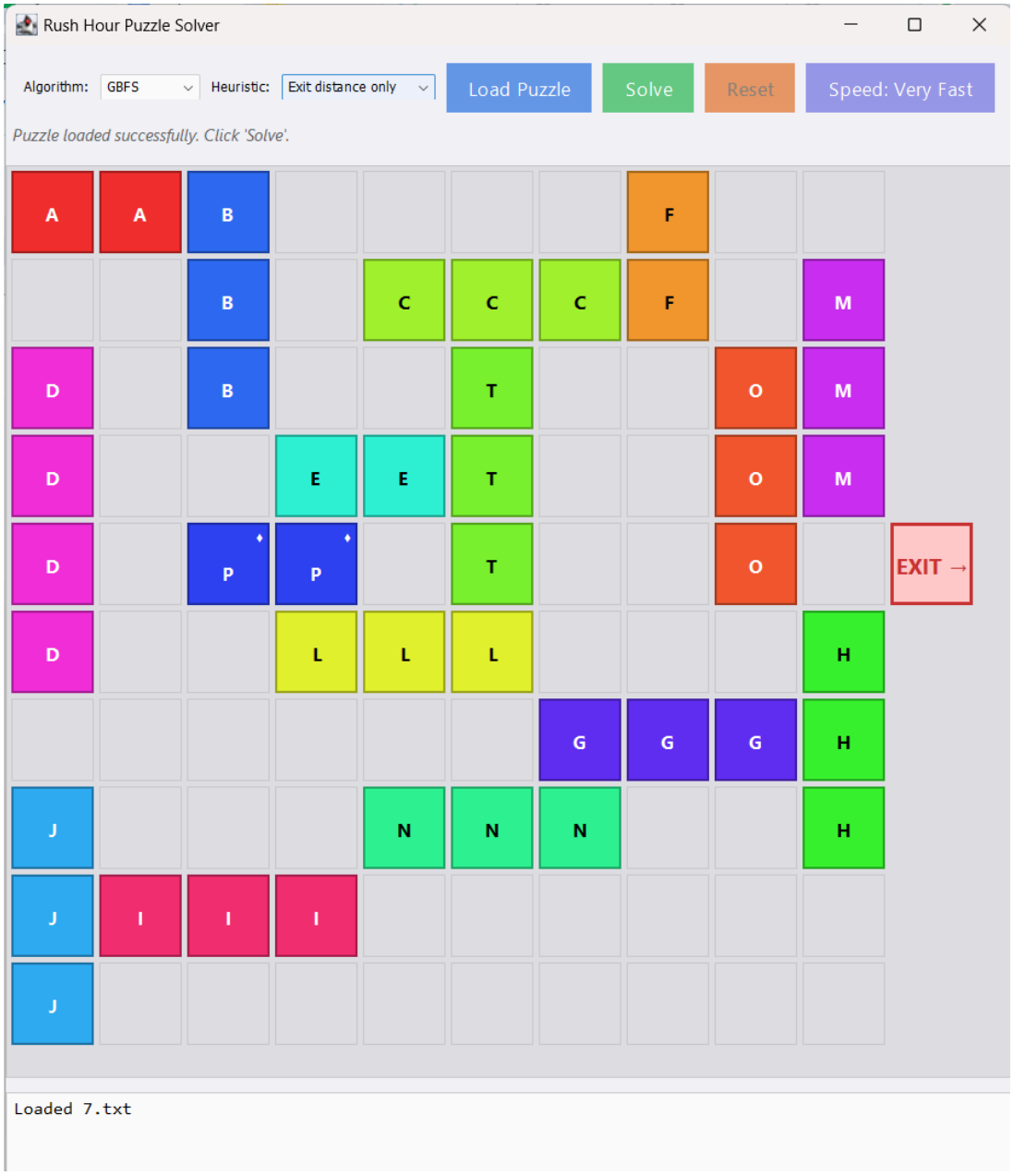
Input.txt

Output


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
EXIT ↑
A A A A
F B B
F E E
F S S G G
Loaded 4.txt
Solving with GBFS...
Found: true | Nodes: 28
Runtime: 0.348 ms
Initial state
Step 1: Move car B ← left
Step 2: Move car S → right
Step 3: Move car F ↓ down

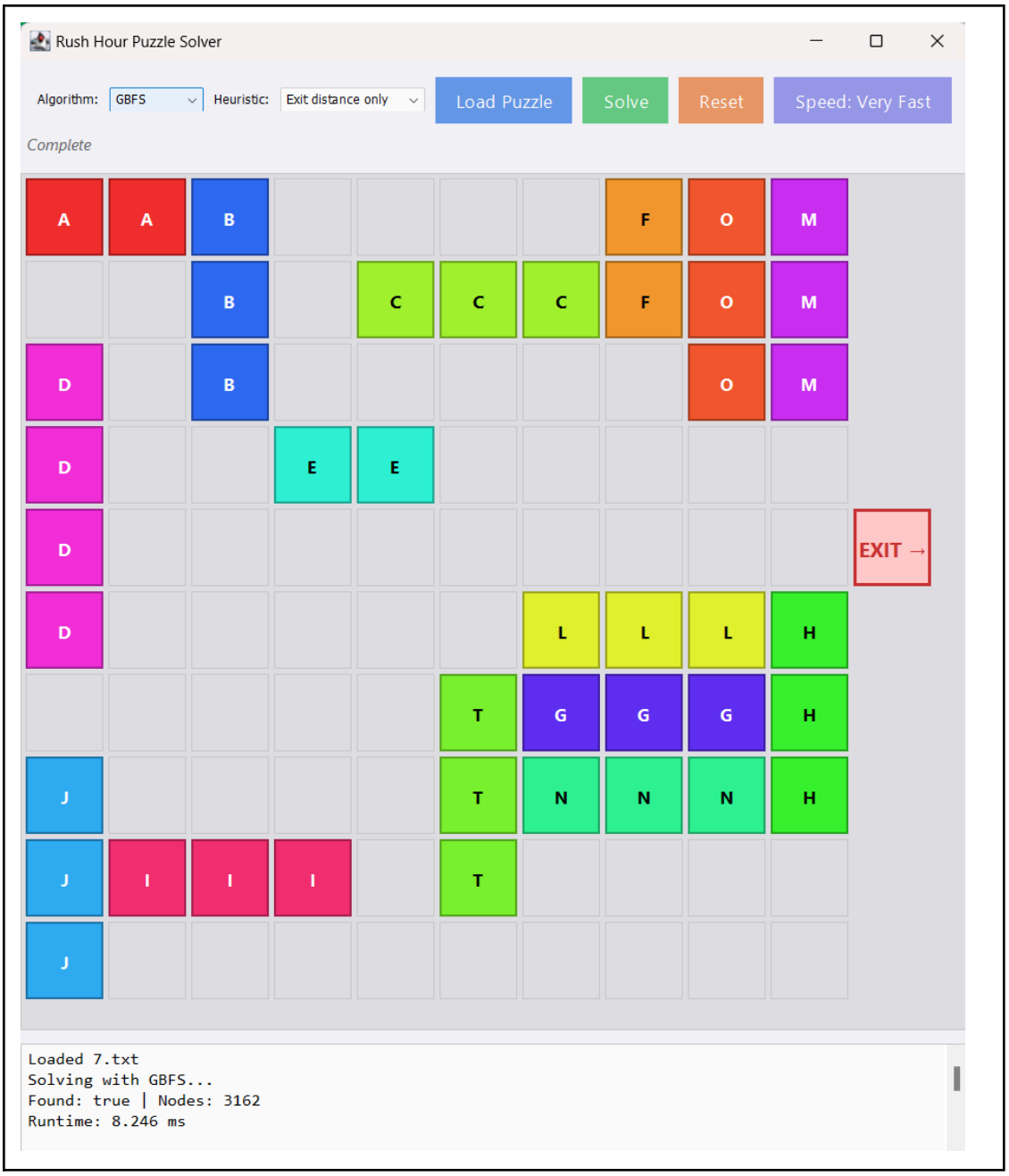


Test Case 4

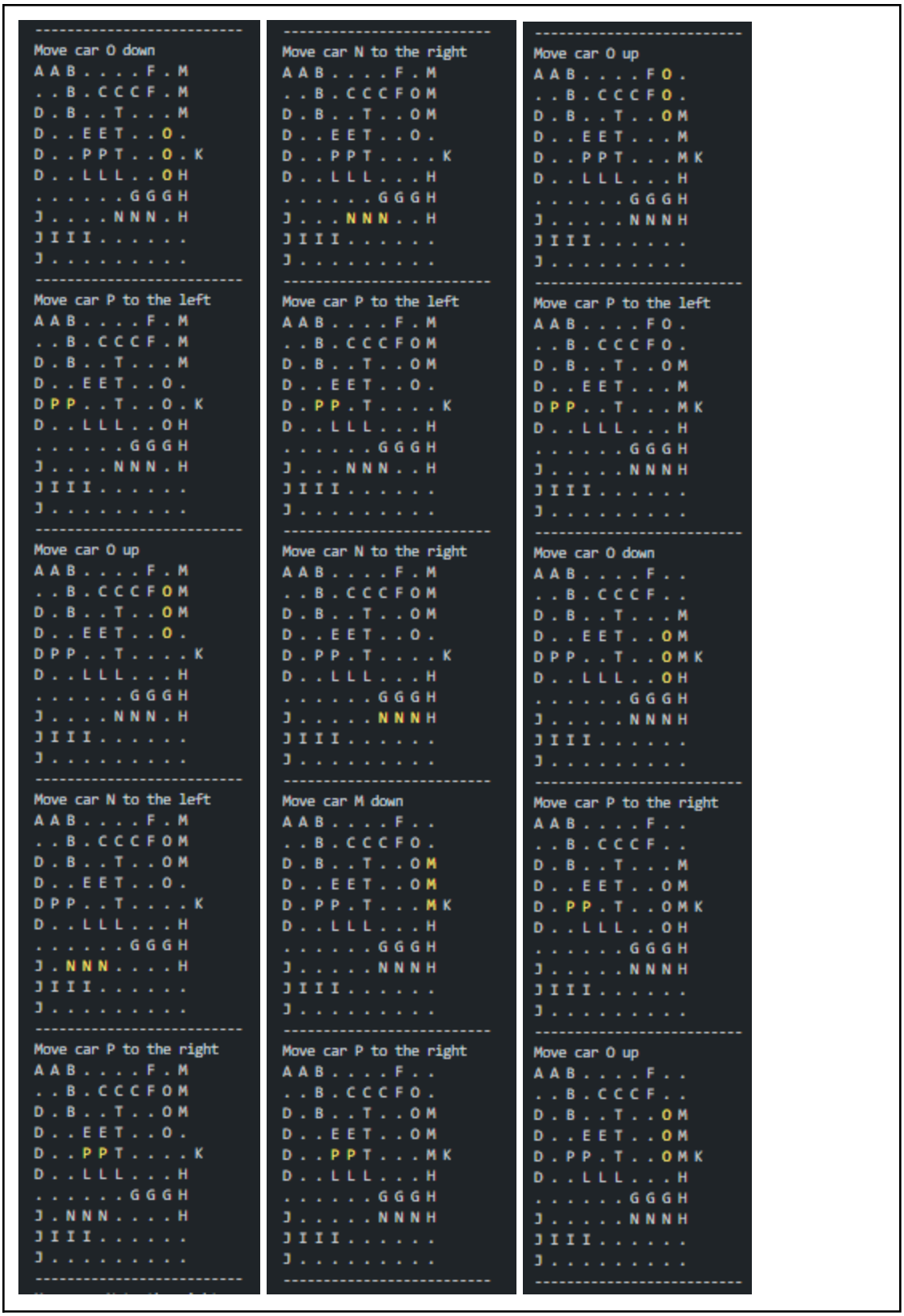
Input.txt

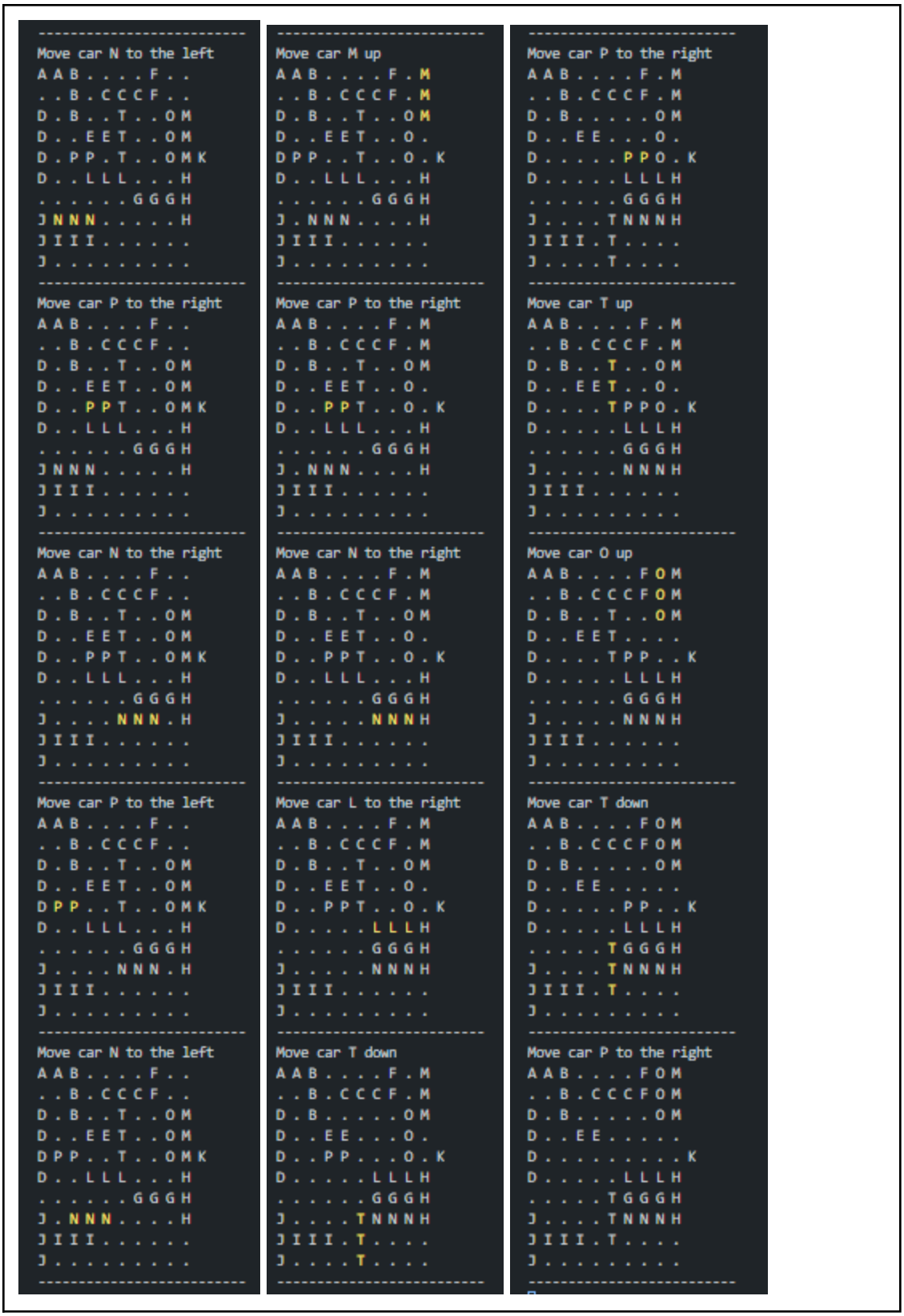

Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
EXIT →
Loaded 7.txt

Output


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Exit distance only
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
EXIT →
Loaded 7.txt
Solving with GBFS...
Found: true | Nodes: 3162
Runtime: 8.246 ms

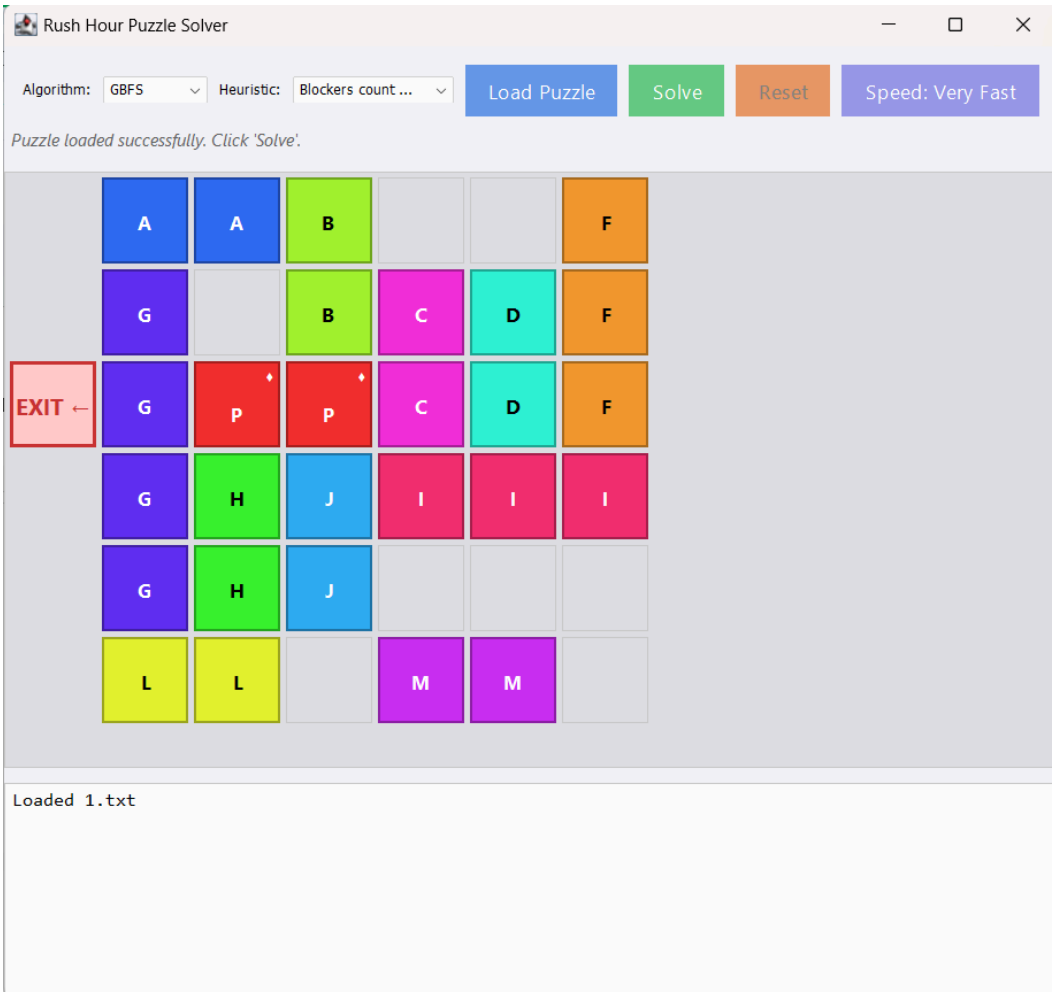

generating solution....
Exit found
Move car P to the right
A A B . . . . F . .
. . B . C C C F . M
D . B . . T . . O M
D . . E E T . . O M
D . . P P T . . O . K
D . . L L L . . . H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car O up
A A B . . . . F O .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . M
D . . P P T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car P to the left
A A B . . . . F O .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . M
D P P . . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car O down
A A B . . . . F . .
. . B . C C C F . M
D . B . . T . . . M
D . . E E T . . O M
D P P . . T . . O . K
D . . L L L . . O H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A B . . . . F . .
. . B . C C C F . M
D . B . . T . . . M
D . . E E T . . O M
D . P P . T . . O . K
D . . L L L . . O H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car N to the right
A A B . . . . F . .
. . B . C C C F . M
D . B . . T . . . M
D . . E E T . . O M
D . P P . T . . O . K
D . . L L L . . O H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A B . . . . F . .
. . B . C C C F . M
D . B . . T . . . M
D . . E E T . . O M
D . . P P T . . O . K
D . . L L L . . O H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car O up
A A B . . . . F . .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O M
D . . P P T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car P to the left
A A B . . . . F . .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O M
D P P . . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car N to the left
A A B . . . . F . .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O M
D P P . . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J N N N . . . . . H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A B . . . . F . .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O M
D . P P . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J N N N . . . . . H
J I I I . . . . . .
J . . . . . .
Move car O up
A A B . . . . F O .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . M
D . P P . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J N N N . . . . . H
J I I I . . . . . .
J . . . . . .
Move car N to the right
A A B . . . . F O .
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . M
D . P P . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . . . .
Move car M up
A A B . . . . F O M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . .
D . P P . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A B . . . . F O M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . .
D . . P P T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . . . .

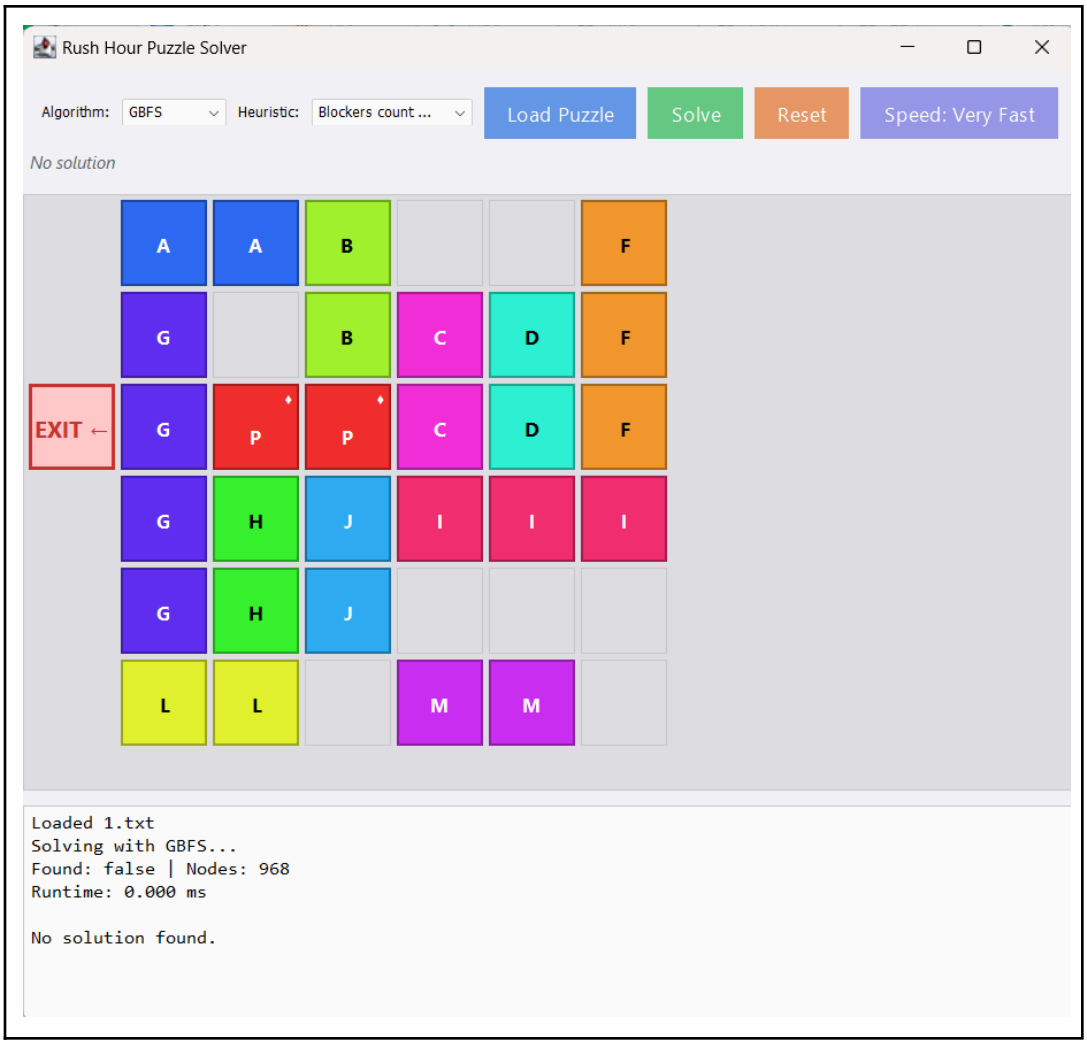

Move car O down
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . . M
D . . E E T . . O .
D . . P P T . . O . K
D . . L L L . . O H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . .
Move car P to the left
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . . M
D . . E E T . . O .
D P P . . T . . O . K
D . . L L L . . O H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . .
Move car O up
A A B . . . . F . M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O .
D P P . . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . .
Move car N to the left
A A B . . . . F . M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O .
D P P . . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . N N N . . . . H
J I I I . . . . . .
J . . . .
Move car P to the right
A A B . . . . F . M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O .
D . . P P T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . N N N . . . . H
J I I I . . . . . .
J . . . .
Move car N to the right
A A B . . . . F . M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O .
D . . P P T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . .
Move car P to the left
A A B . . . . F . M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O .
D . P P . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . .
Move car N to the right
A A B . . . . F . M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . O .
D . P P . T . . . . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car M down
A A B . . . . F . .
. . B . C C C F O .
D . B . . T . . O M
D . . E E T . . O M
D . P P . T . . . M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car P to the right
A A B . . . . F . .
. . B . C C C F O .
D . B . . T . . O M
D . . E E T . . O M
D . . P P T . . . M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car O up
A A B . . . . F O .
. . B . C C C F O .
D . B . . T . . O M
D . . E E T . . . M
D . . P P T . . . M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car P to the left
A A B . . . . F O .
. . B . C C C F O .
D . B . . T . . O M
D . . E E T . . . M
D P P . . T . . . M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car O down
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . . M
D . . E E T . . O M
D P P . . T . . O M K
D . . L L L . . O H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car P to the right
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . . M
D . . E E T . . O M
D . P P . T . . O M K
D . . L L L . . O H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .
Move car O up
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . O M
D . . E E T . . O M
D . P P . T . . O M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . .


Move car N to the left
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . O M
D . . E E T . . O M
D . P P . T . . O M K
D . . L L L . . . H
. . . . . . G G G H
J N N N . . . . . H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . O M
D . . E E T . . O M
D . . P P T . . O M K
D . . L L L . . . H
. . . . . . G G G H
J N N N . . . . . H
J I I I . . . . . .
J . . . . . .
Move car N to the right
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . O M
D . . E E T . . O M
D . . P P T . . O M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . . . .
Move car P to the left
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . O M
D . . E E T . . O M
D P P . . T . . O M K
D . . L L L . . . H
. . . . . . G G G H
J . . . . N N N . H
J I I I . . . . . .
J . . . . . .
Move car N to the left
A A B . . . . F . .
. . B . C C C F . .
D . B . . T . . O M
D . . E E T . . O M
D P P . . T . . O M K
D . . L L L . . . H
. . . . . . G G G H
J . N N N . . . . H
J I I I . . . . . .
J . . . . . .
Move car M up
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . . E E T . . O .
D P P . . T . . O . K
D . . L L L . . . H
. . . . . . G G G H
J . N N N . . . . H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . . E E T . . O .
D . . P P T . . O . K
D . . L L L . . . H
. . . . . . G G G H
J . N N N . . . . H
J I I I . . . . . .
J . . . . . .
Move car N to the right
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . . E E T . . O .
D . . P P T . . O . K
D . . L L L . . . H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car L to the right
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . . E E T . . O .
D . . P P T . . O . K
D . . . . . L L L H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car T down
A A B . . . . F . M
. . B . C C C F . M
D . B . . . . . O M
D . . E E . . . O .
D . . P P . . . O . K
D . . . . . L L L H
. . . . . . G G G H
J . . . . T N N N H
J I I I . T . . . .
J . . . . T . . . .
Move car P to the right
A A B . . . . F . M
. . B . C C C F . M
D . B . . . . . O M
D . . E E . . . O .
D . . . . . P P O . K
D . . . . . L L L H
. . . . . . G G G H
J . . . . T N N N H
J I I I . T . . . .
J . . . . T . . . .
Move car T up
A A B . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . . E E T . . O .
D . . . . T P P O . K
D . . . . . L L L H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car O up
A A B . . . . F O M
. . B . C C C F O M
D . B . . T . . O M
D . . E E T . . . .
D . . . . T P P . . K
D . . . . . L L L H
. . . . . . G G G H
J . . . . . N N N H
J I I I . . . . . .
J . . . . . .
Move car T down
A A B . . . . F O M
. . B . C C C F O M
D . B . . . . . O M
D . . E E . . . . .
D . . . . . P P . . K
D . . . . . L L L H
. . . . . T G G G H
J . . . . T N N N H
J I I I . T . . . .
J . . . . . .
Move car P to the right
A A B . . . . F O M
. . B . C C C F O M
D . B . . . . . O M
D . . E E . . . . .
D . . . . K
D . . . . . L L L H
. . . . . T G G G H
J . . . . T N N N H
J I I I . T . . . .
J . . . . . .

3.3.2.2 Blockers Count Heuristic

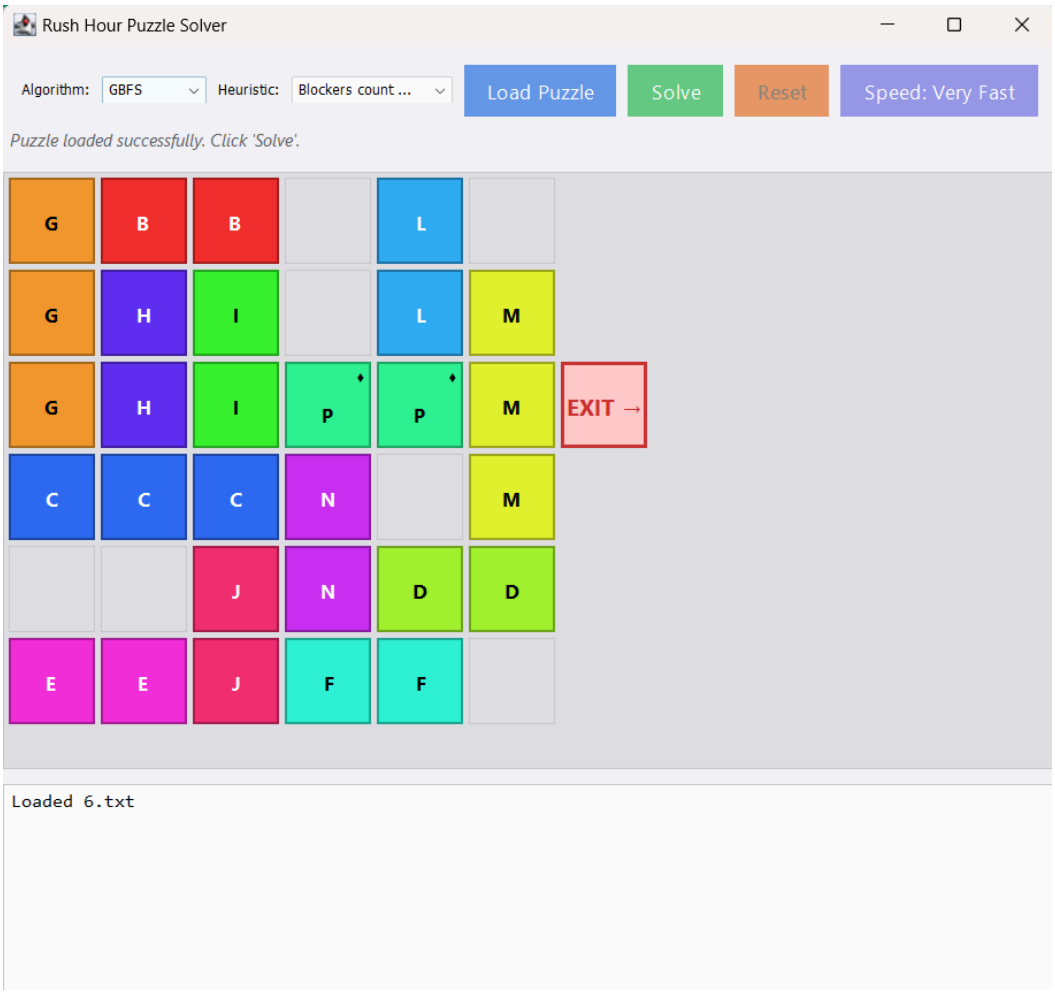
Test Case 1

Input.txt

Output

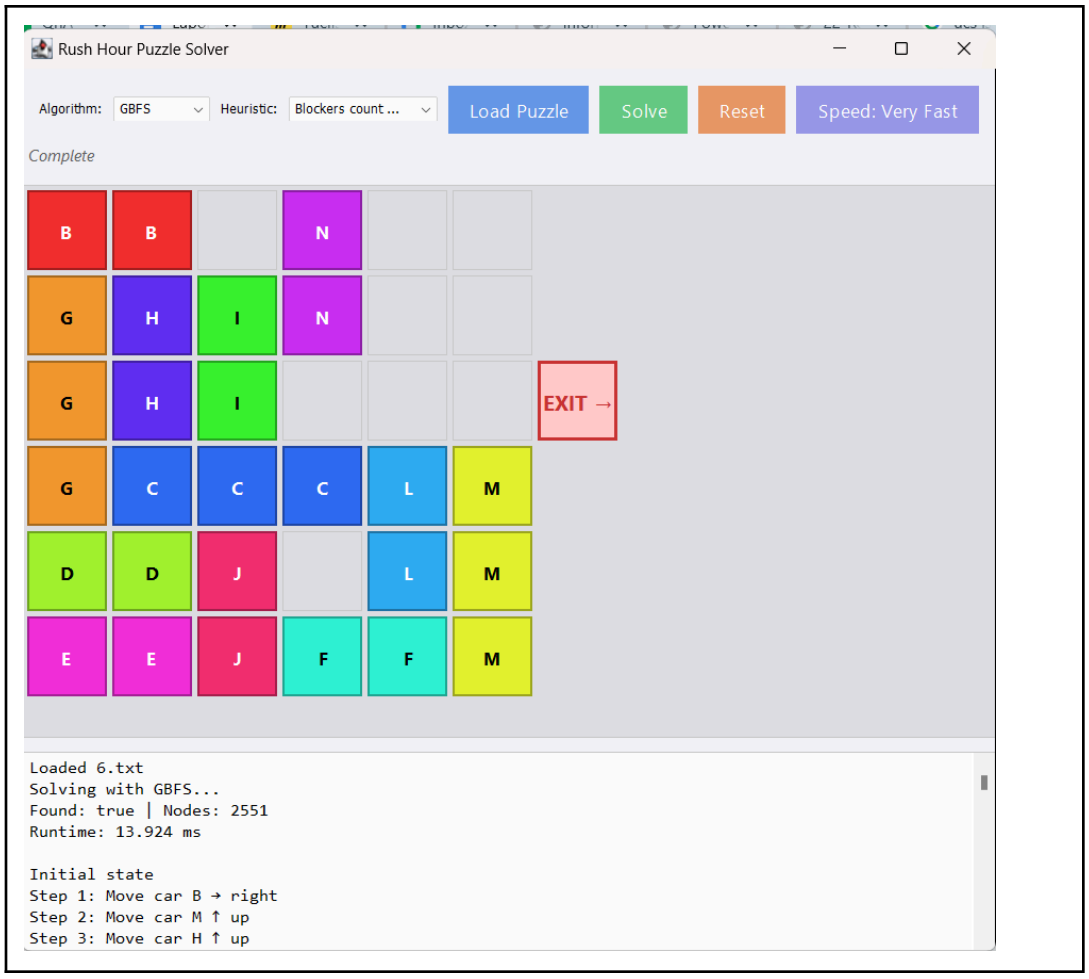


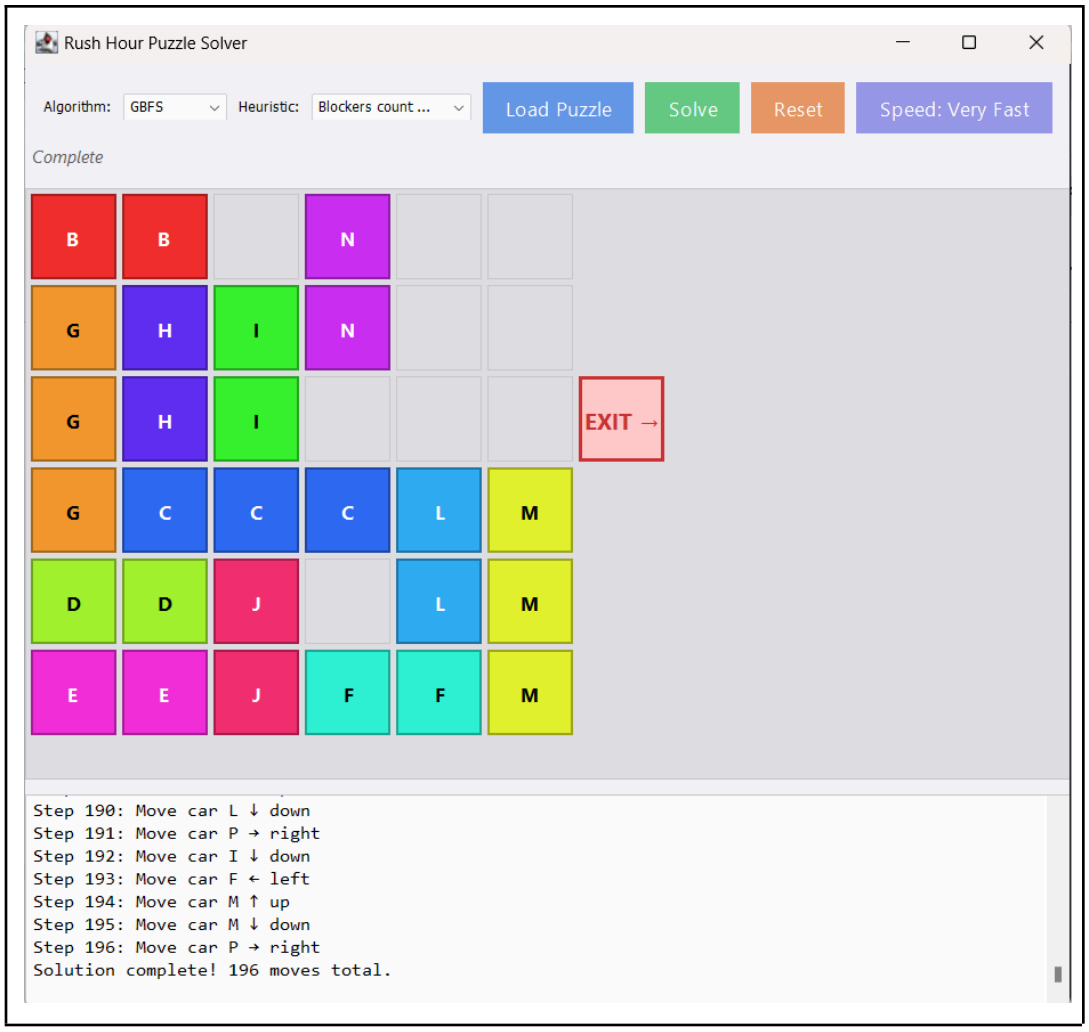
Test Case 2

Input.txt


Rush Hour Puzzle Solver
Algorithm:
GBFS
Heuristic:
Blockers count ...
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
G B B L
G H I L M
G H I P P M EXIT →
C C C N M
J N D D
E E J F F
Loaded 6.txt

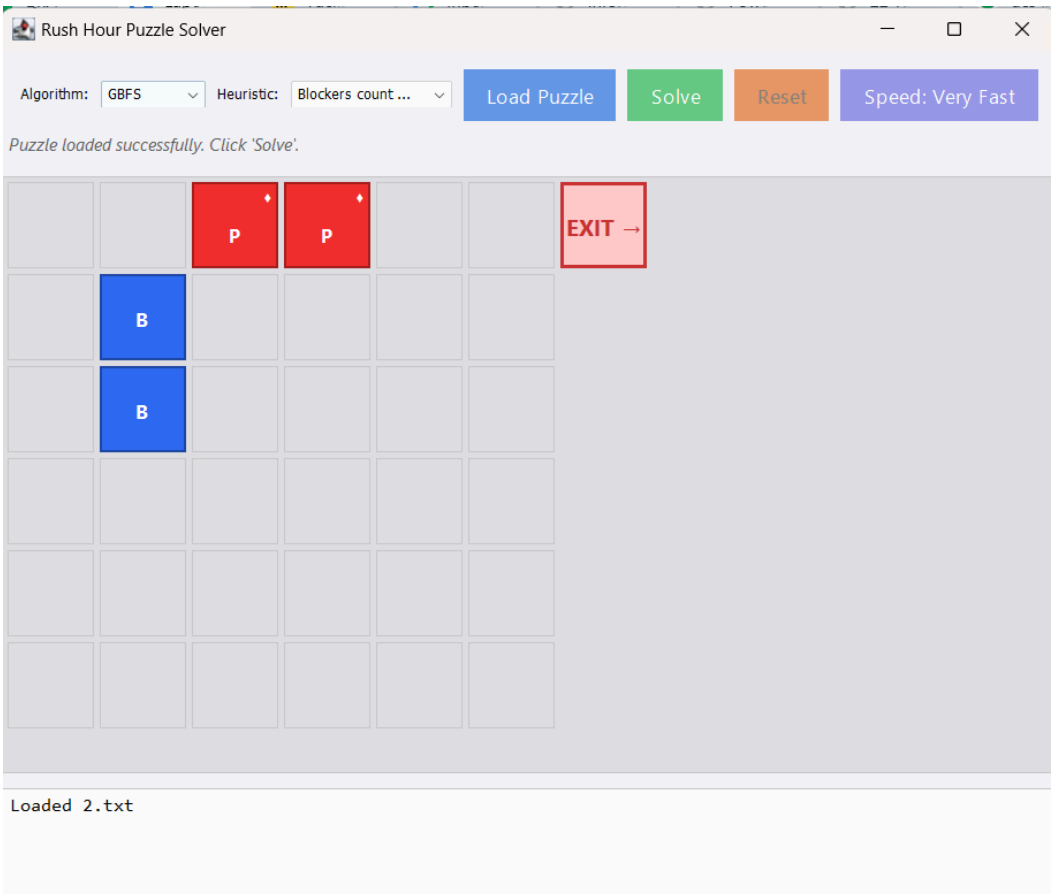
Output


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Blockers count ...
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
EXIT →
Loaded 6.txt
Solving with GBFS...
Found: true | Nodes: 2551
Runtime: 13.924 ms
Initial state
Step 1: Move car B → right
Step 2: Move car M ↑ up
Step 3: Move car H ↑ up

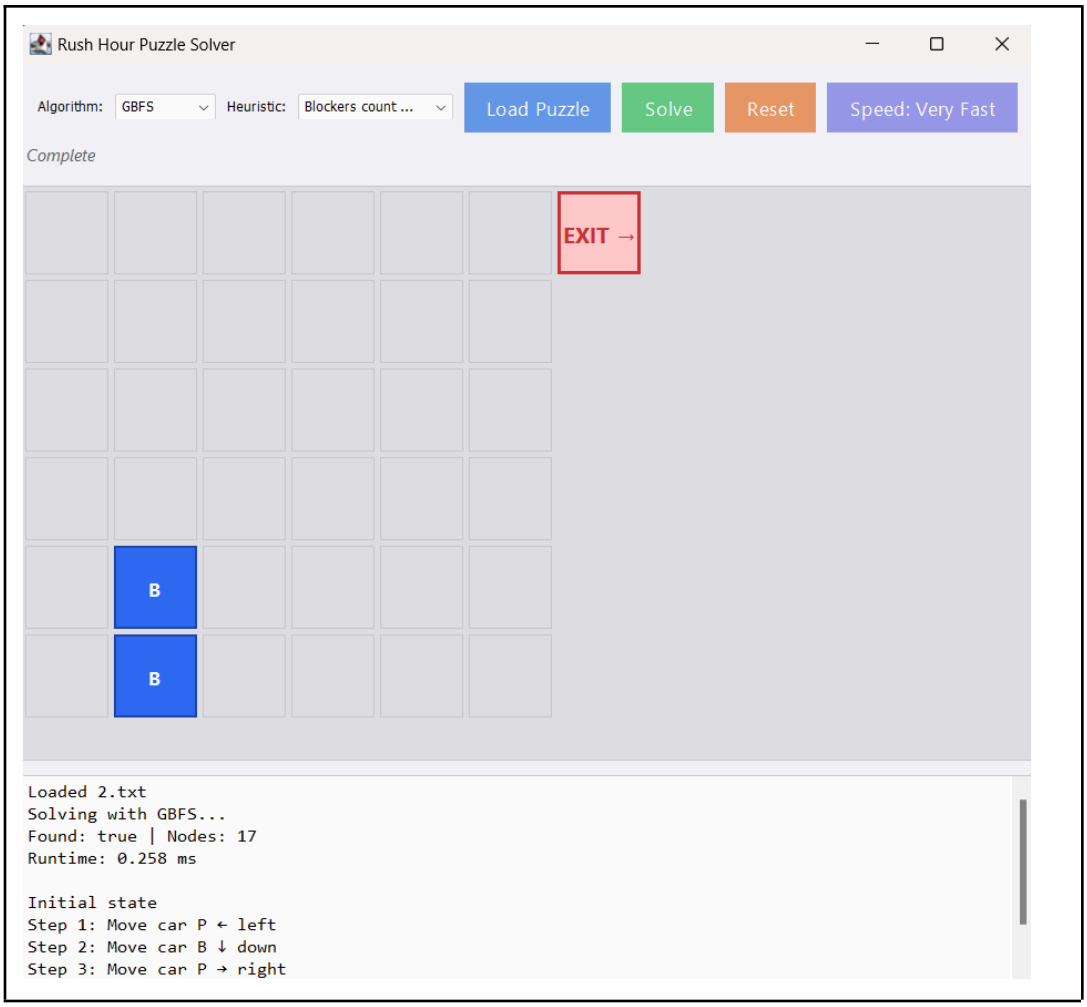


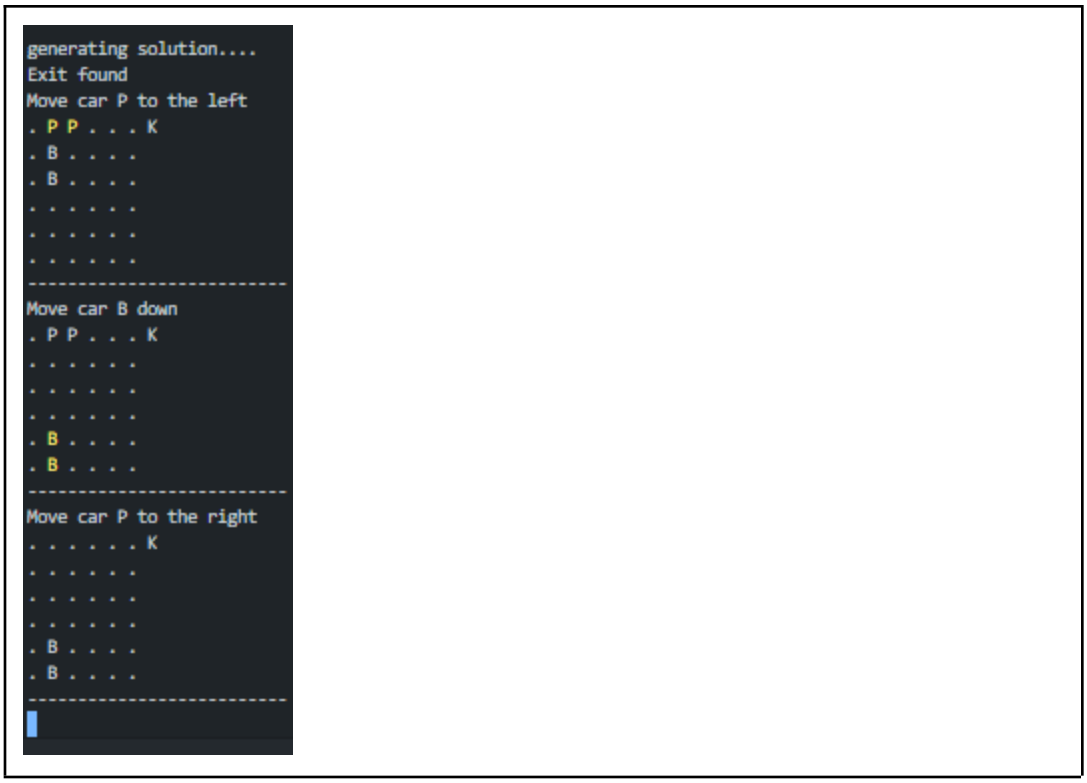
Test Case 3

Input.txt


Rush Hour Puzzle Solver
Algorithm:
GBFS
Heuristic:
Blockers count ...
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
P
P
EXIT →
B
B
Loaded 2.txt

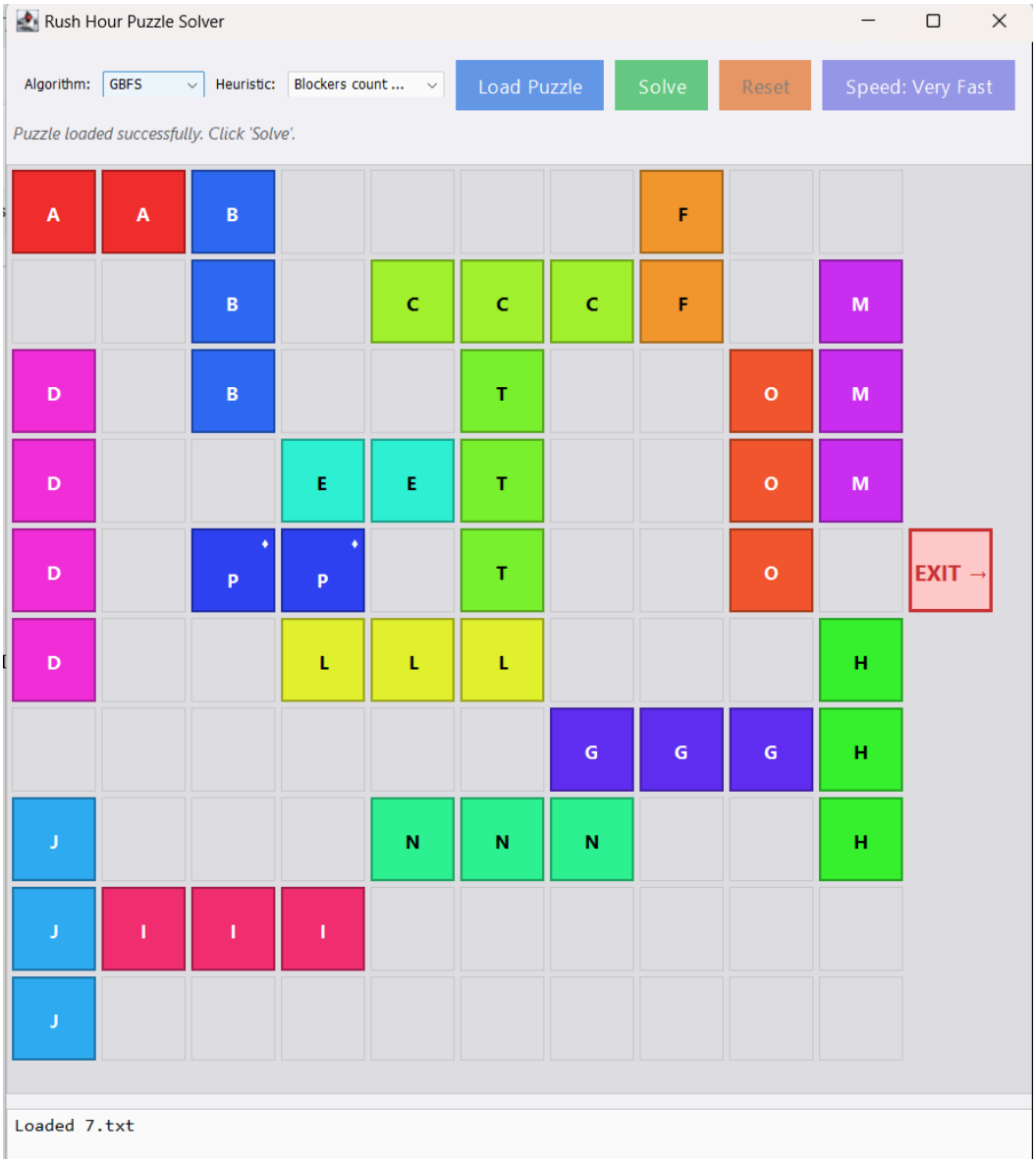
Output


Rush Hour Puzzle Solver
Algorithm:
GBFS
Heuristic:
Blockers count ...
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
EXIT →
B
B
Loaded 2.txt
Solving with GBFS...
Found: true | Nodes: 17
Runtime: 0.258 ms
Initial state
Step 1: Move car P ← left
Step 2: Move car B ↓ down
Step 3: Move car P → right

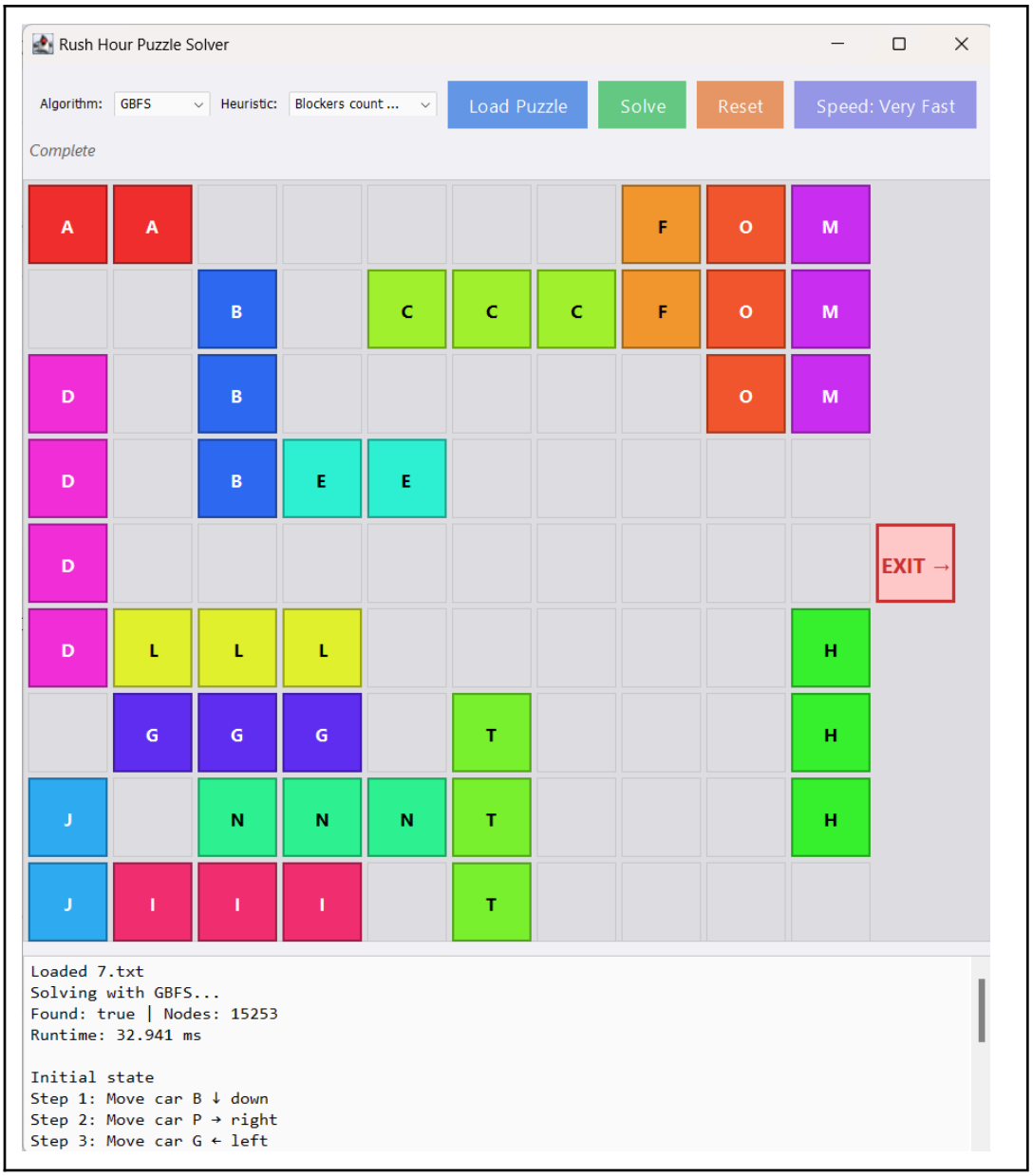


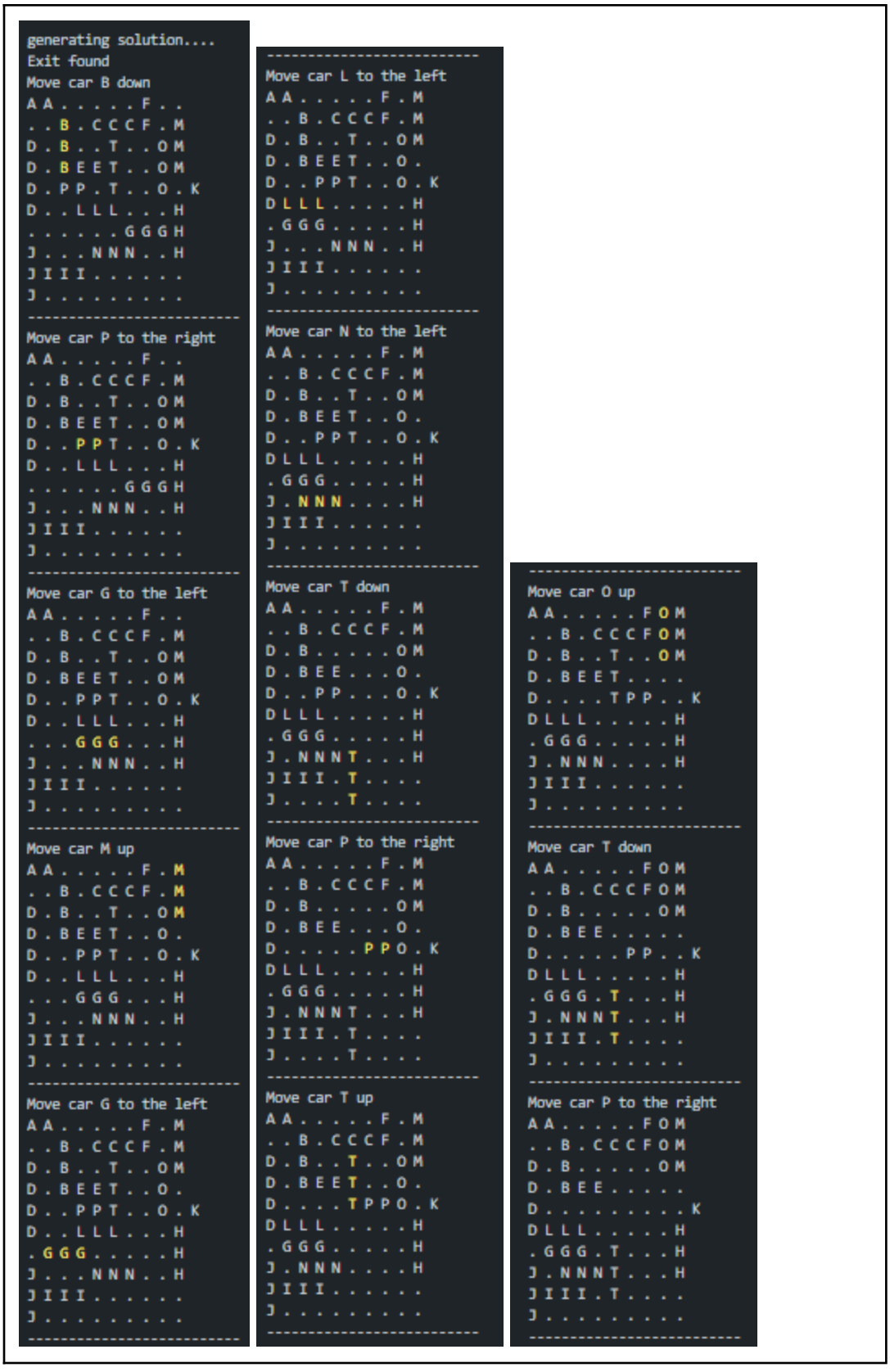
Test Case 4

Input.txt


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Blockers count ...
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
EXIT →
Loaded 7.txt

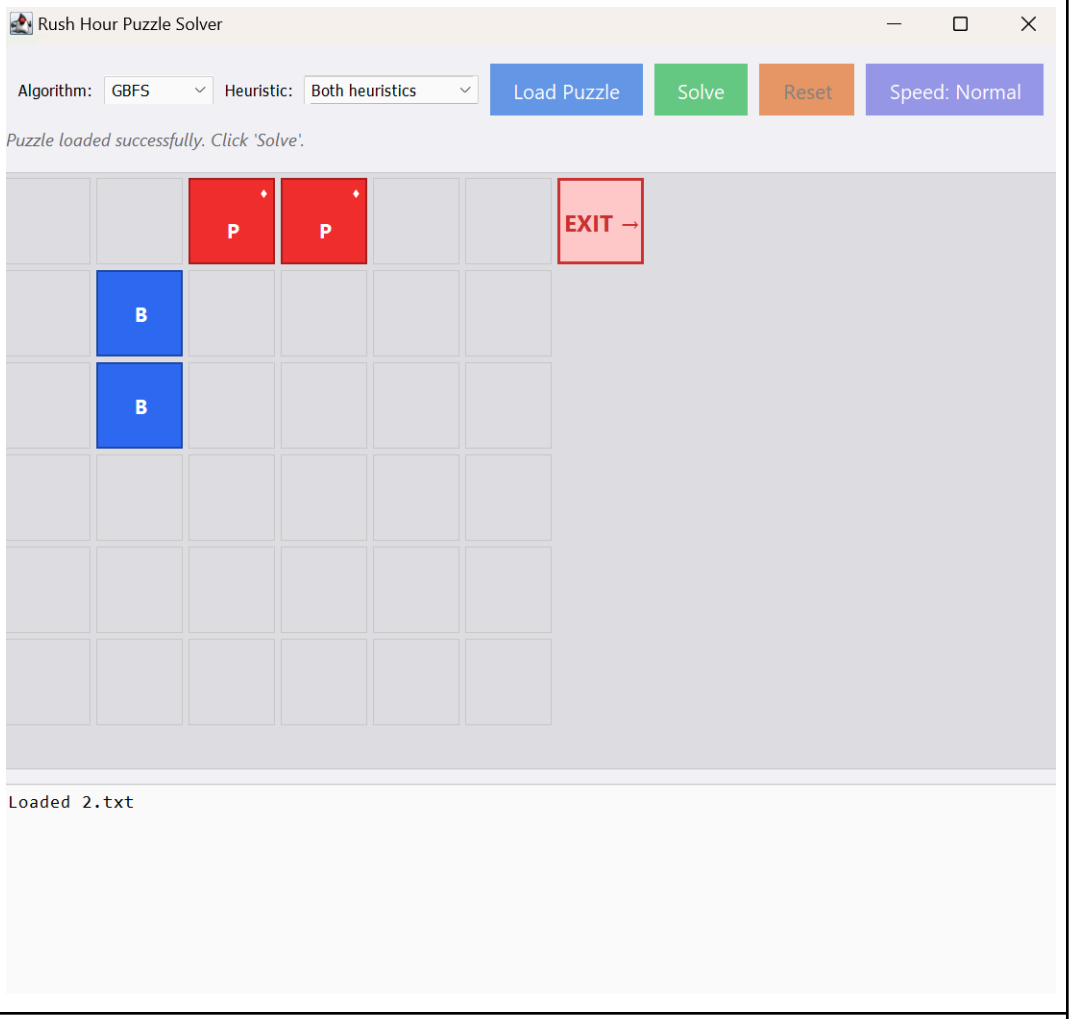
Output

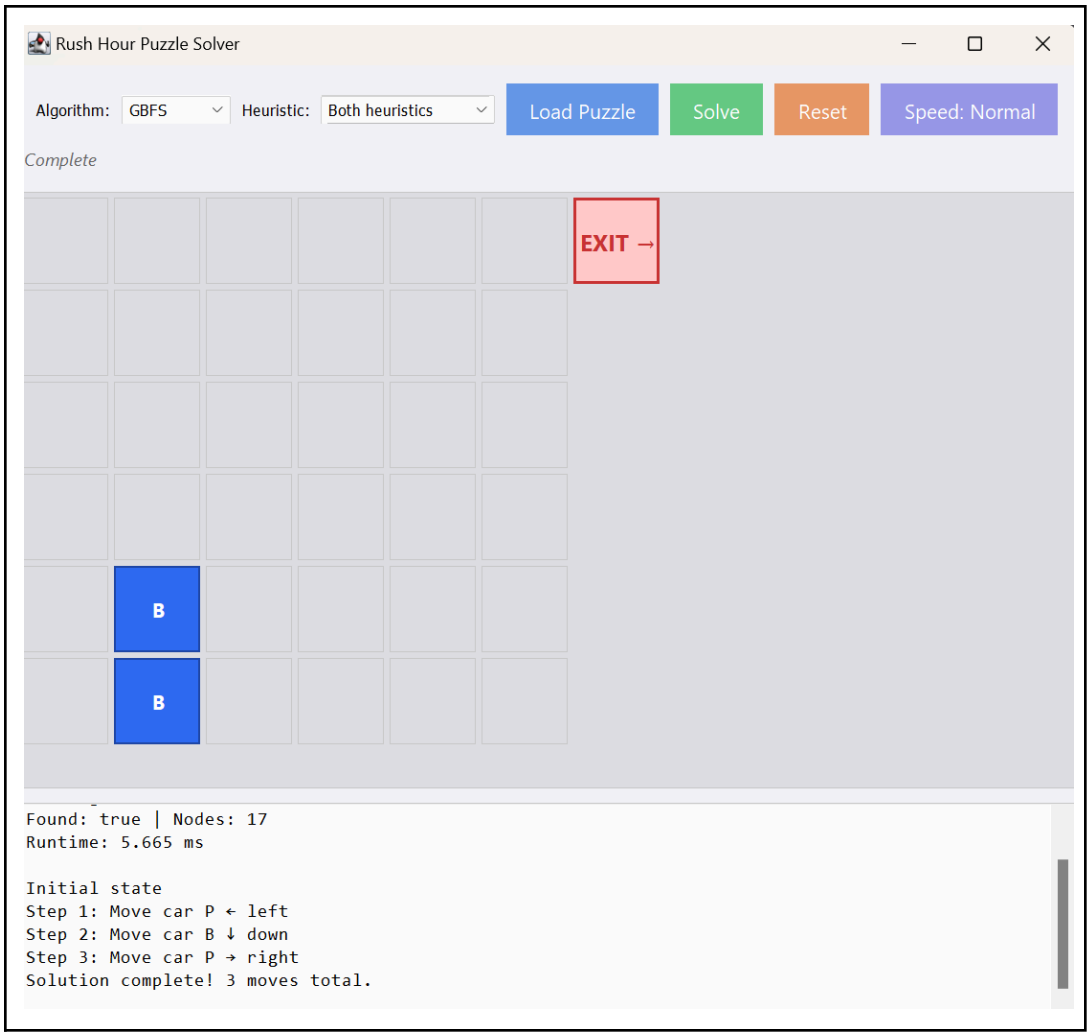

Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Blockers count ...
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
EXIT →
Loaded 7.txt
Solving with GBFS...
Found: true | Nodes: 15253
Runtime: 32.941 ms
Initial state
Step 1: Move car B ↓ down
Step 2: Move car P → right
Step 3: Move car G ← left


generating solution....
Exit found
Move car B down
A A . . . . . F . .
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O M
D . P P . T . . O . K
D . . L L L . . . H
. . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car P to the right
A A . . . . . F . .
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O M
D . . P P T . . O . K
D . . L L L . . . H
. . . . . G G G H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car G to the left
A A . . . . . F . .
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O M
D . . P P T . . O . K
D . . L L L . . . H
. . . G G G . . . H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car M up
A A . . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O .
D . . P P T . . O . K
D . . L L L . . . H
. . . G G G . . . H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car G to the left
A A . . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O .
D . . P P T . . O . K
D . . L L L . . . H
. G G G . . . . . H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car L to the left
A A . . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O .
D . . P P T . . O . K
D L L L . . . . . H
. G G G . . . . . H
J . . . N N N . . H
J I I I . . . . . .
J . . . . . .
Move car N to the left
A A . . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O .
D . . P P T . . O . K
D L L L . . . . . H
. G G G . . . . . H
J . N N N . . . . H
J I I I . . . . . .
J . . . . . .
Move car T down
A A . . . . . F . M
. . B . C C C F . M
D . B . . . . . O M
D . B E E . . . O .
D . . P P . . . O . K
D L L L . . . . . H
. G G G . . . . . H
J . N N N T . . . H
J I I I . T . . . .
J . . . . T . . . .
Move car P to the right
A A . . . . . F . M
. . B . C C C F . M
D . B . . . . . O M
D . B E E . . . O .
D . . . . . P P O . K
D L L L . . . . . H
. G G G . . . . . H
J . N N N T . . . H
J I I I . T . . . .
J . . . . T . . . .
Move car T up
A A . . . . . F . M
. . B . C C C F . M
D . B . . T . . O M
D . B E E T . . O .
D . . . . T P P O . K
D L L L . . . . . H
. G G G . . . . . H
J . N N N . . . . H
J I I I . . . . . .
J . . . . . .
Move car O up
A A . . . . . F O M
. . B . C C C F O M
D . B . . T . . O M
D . B E E T . . . .
D . . . . T P P . . K
D L L L . . . . . H
. G G G . . . . . H
J . N N N . . . . H
J I I I . . . . . .
J . . . . . .
Move car T down
A A . . . . . F O M
. . B . C C C F O M
D . B . . . . . O M
D . B E E . . . . .
D . . . . . P P . . K
D L L L . . . . . H
. G G G . T . . . H
J . N N N T . . . H
J I I I . T . . . .
J . . . . . .
Move car P to the right
A A . . . . . F O M
. . B . C C C F O M
D . B . . . . . O M
D . B E E . . . . .
D . . . . K
D L L L . . . . . H
. G G G . T . . . H
J . N N N T . . . H
J I I I . T . . . .
J . . . . . .

3.3.2.3 Combined Heuristic (Exit Distance & Blockers Count)

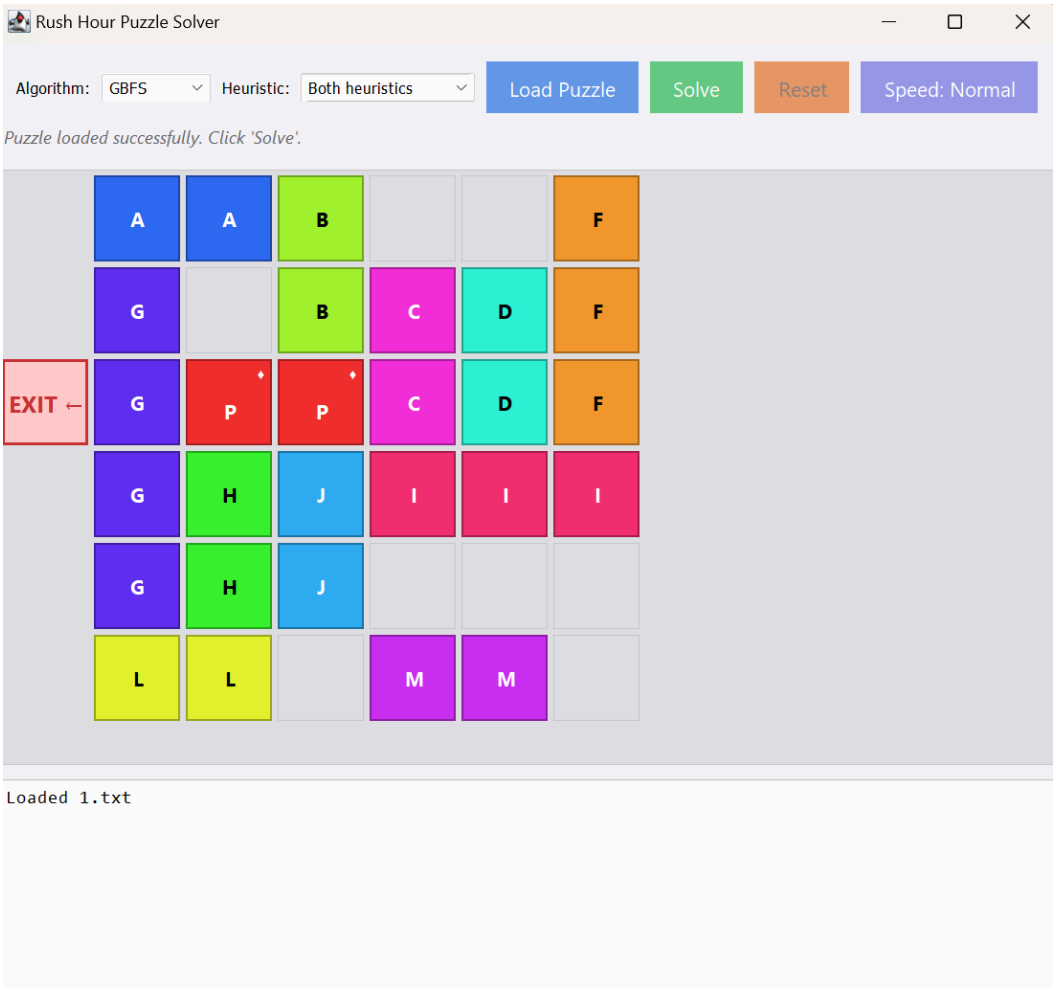
Test Case 1

Input.txt

Output

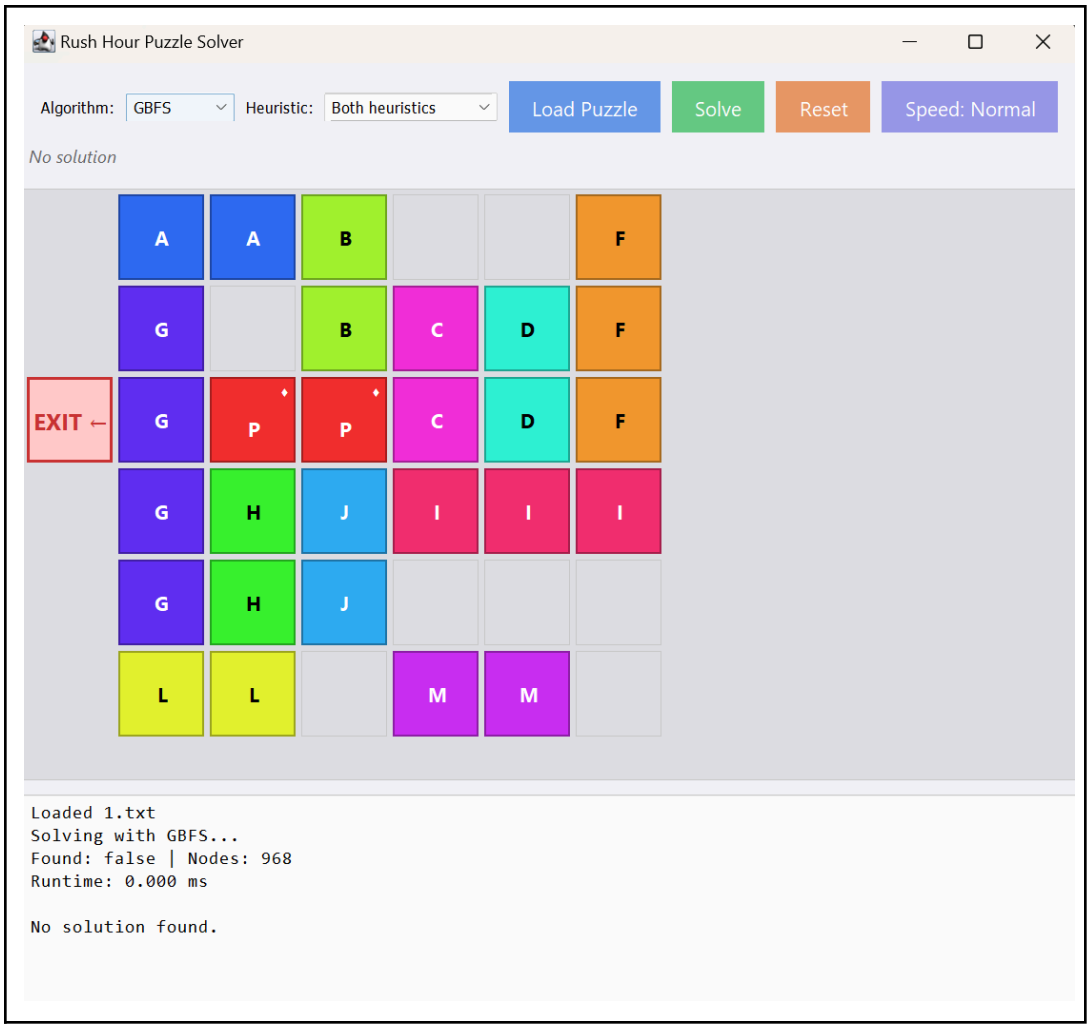


Test Case 2

Input.txt

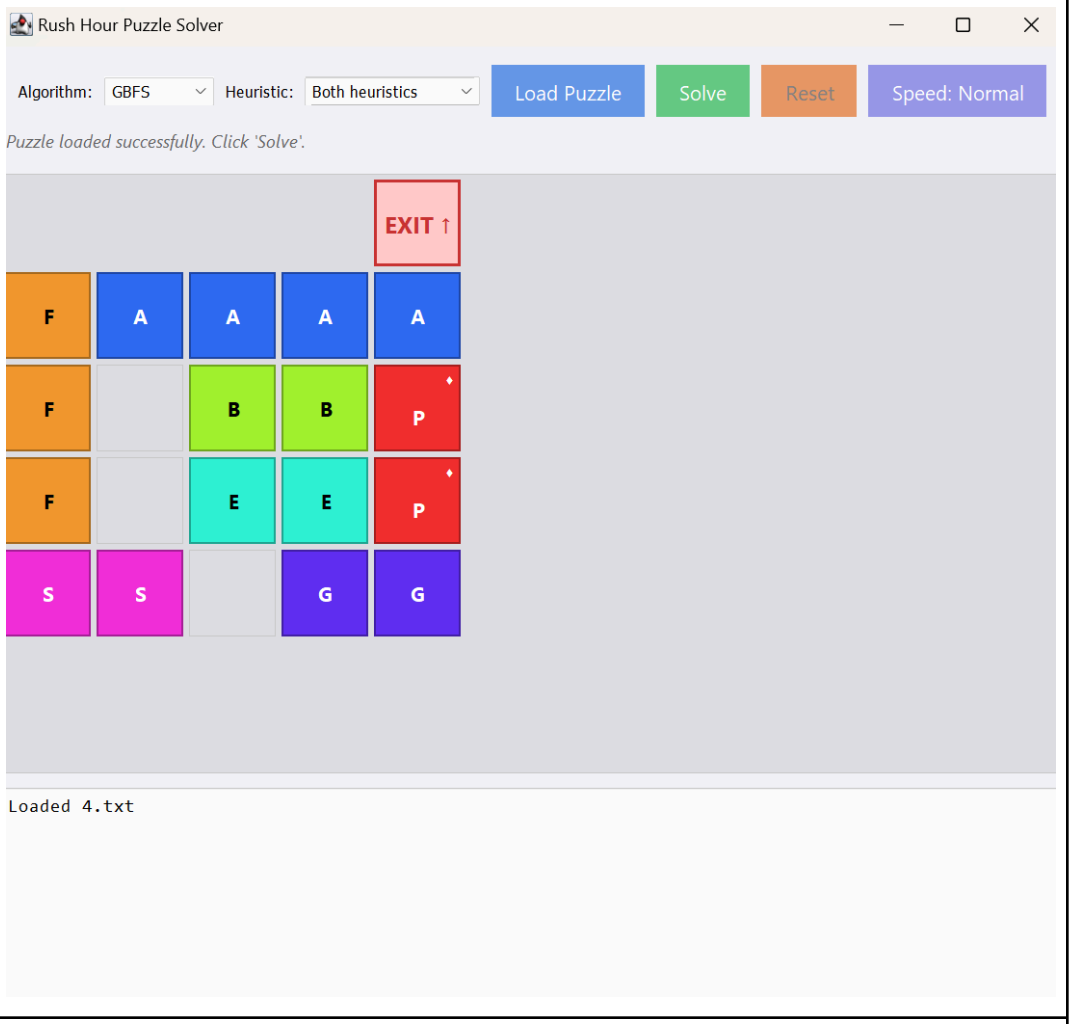

Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Both heuristics
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT ←
Loaded 1.txt

Output

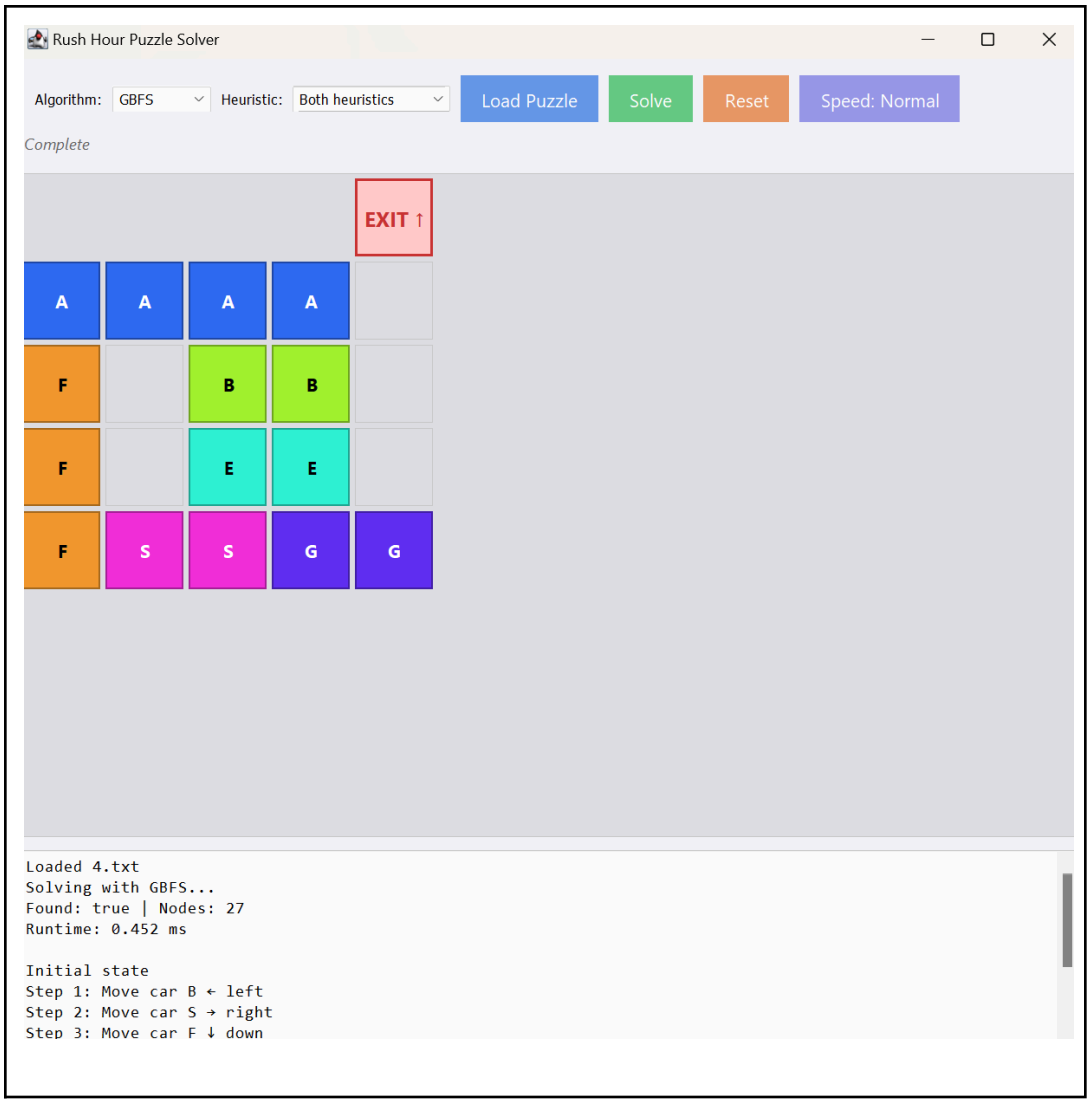


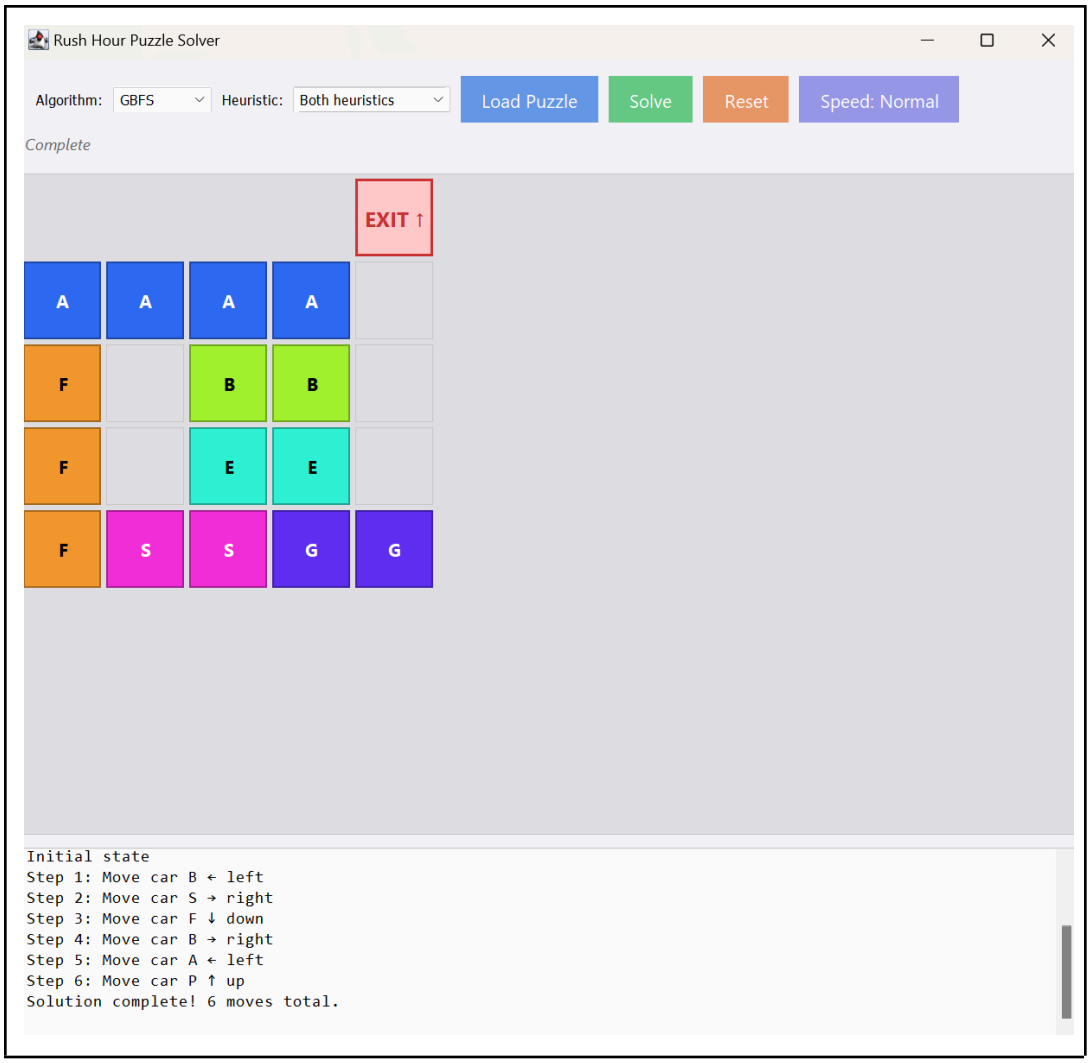
Test Case 3

Input.txt


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Both heuristics
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT ↑
F A A A A
F B B P
F E E P
S S G G
Loaded 4.txt

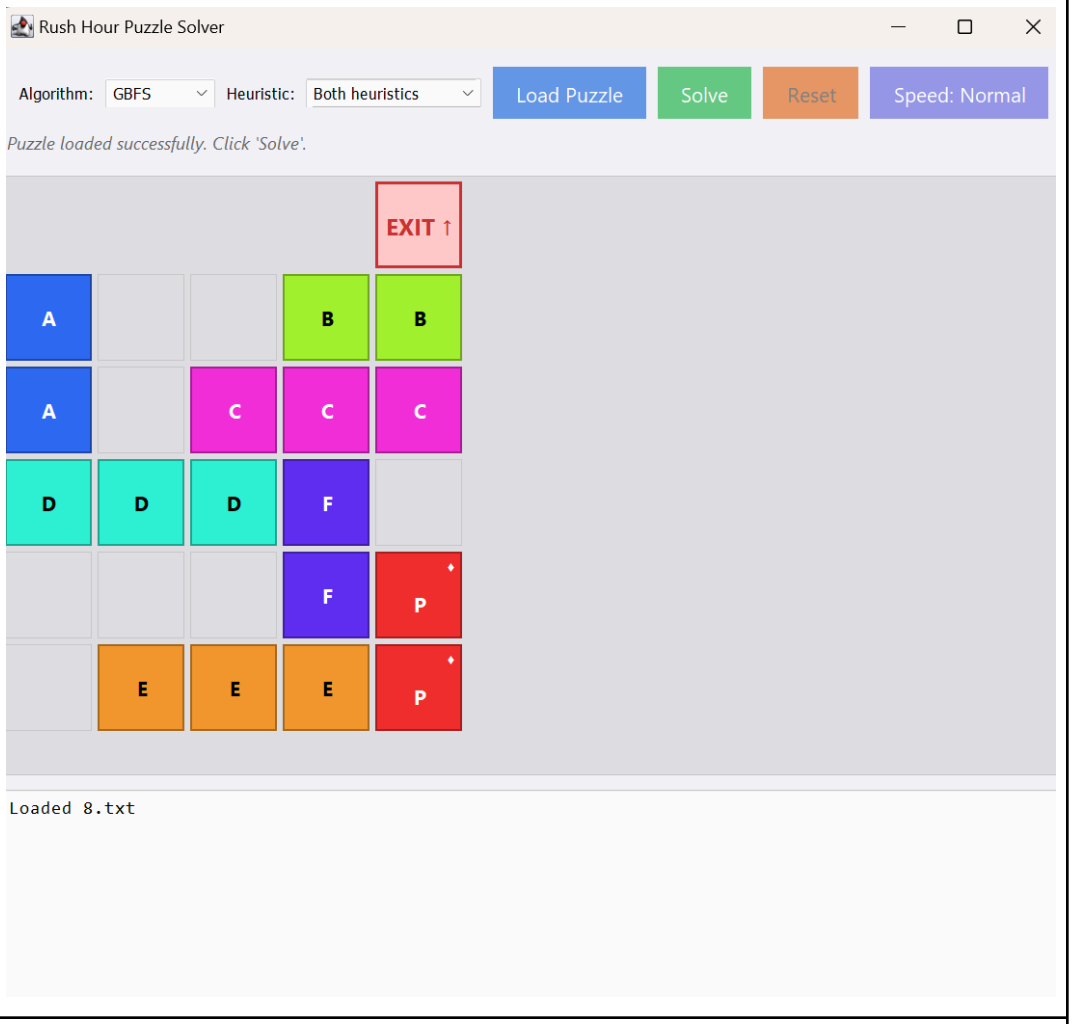
Output


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Both heuristics
Load Puzzle
Solve
Reset
Speed: Normal
Complete
EXIT ↑
A A A A
F B B
F E E
F S S G G
Loaded 4.txt
Solving with GBFS...
Found: true | Nodes: 27
Runtime: 0.452 ms
Initial state
Step 1: Move car B ← left
Step 2: Move car S → right
Step 3: Move car F ↓ down

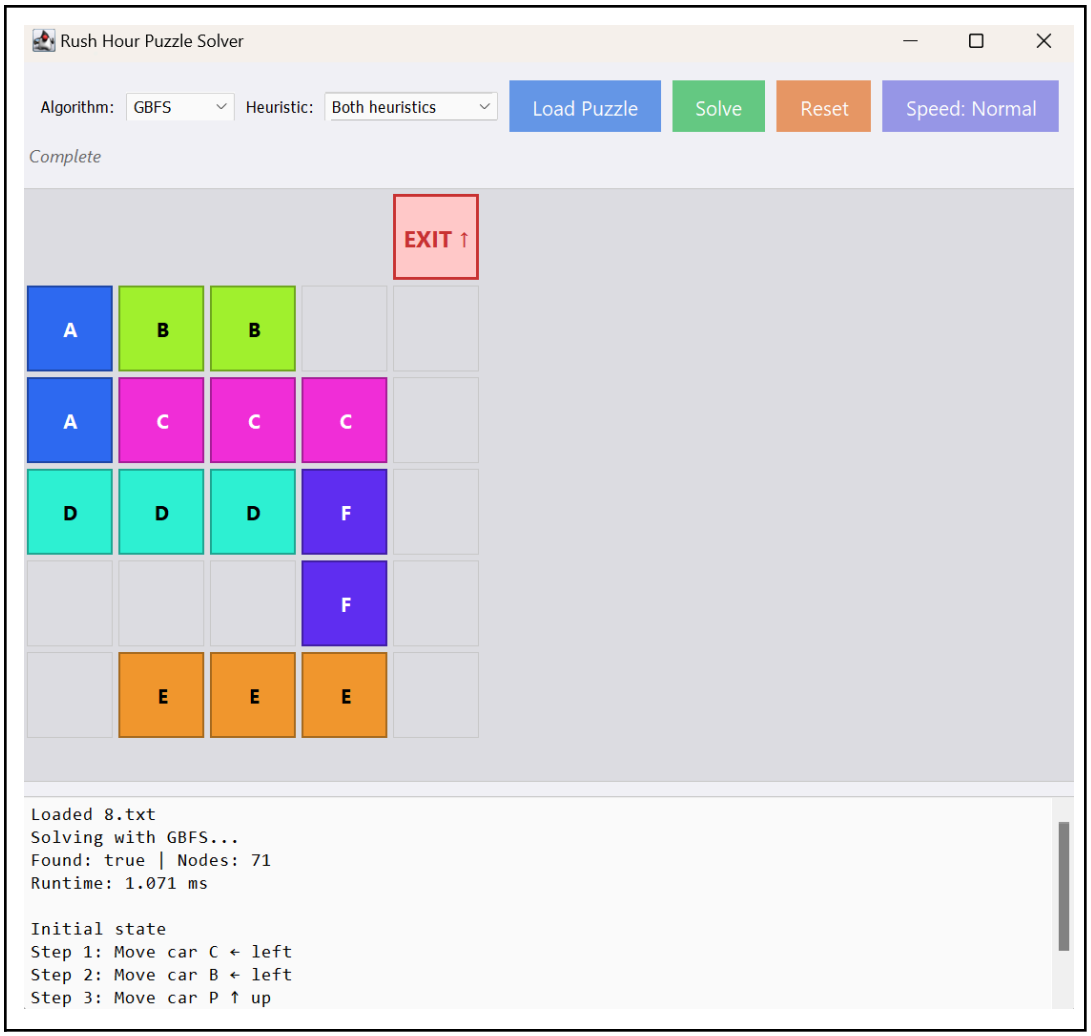


Test Case 4

Input.txt


Rush Hour Puzzle Solver
Algorithm: GBFS
Heuristic: Both heuristics
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT ↑
A
B
B
A
C
C
C
D
D
D
F
F
P
E
E
E
P
Loaded 8.txt

Output

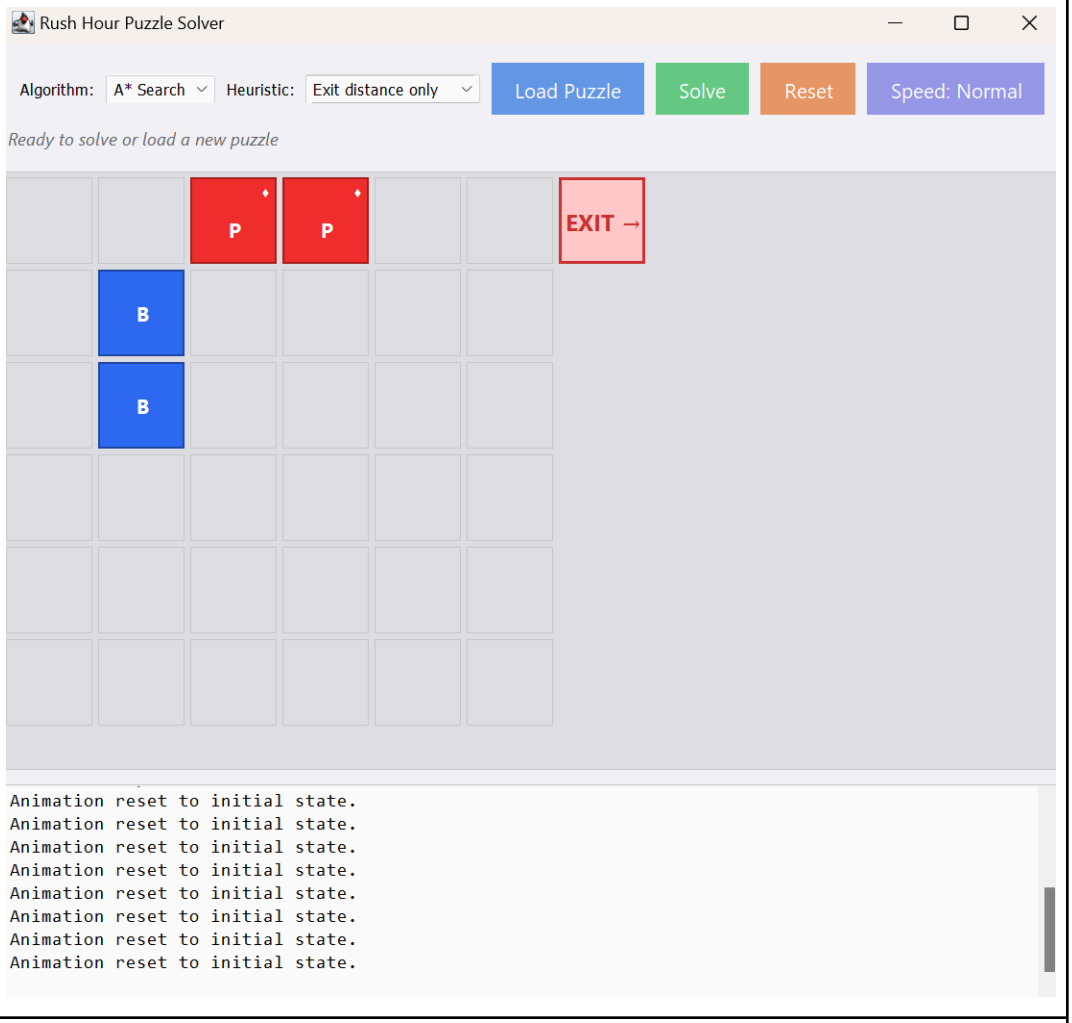


3.3.3 A*

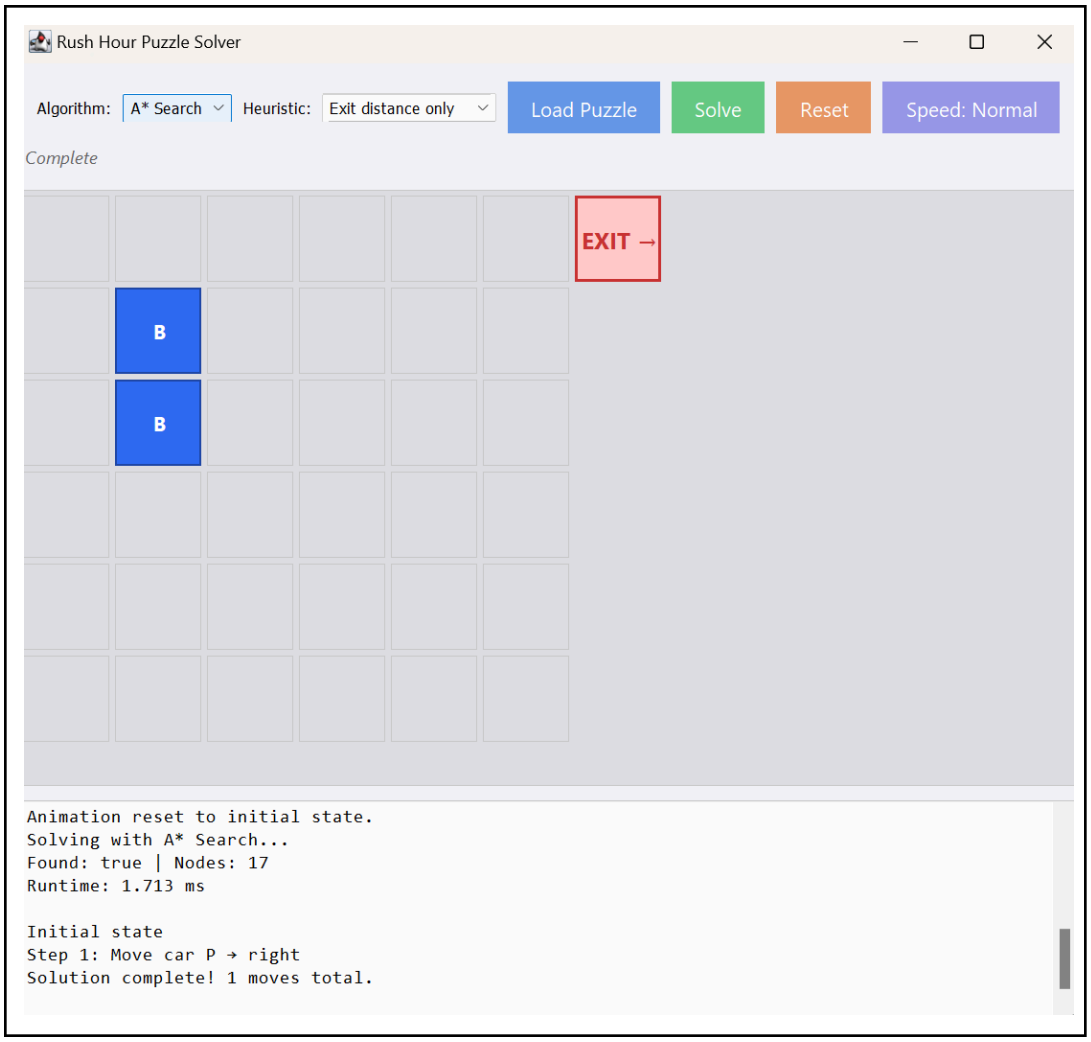
3.3.3.1 Exit Distance Heuristic

Test Case 1

Input.txt

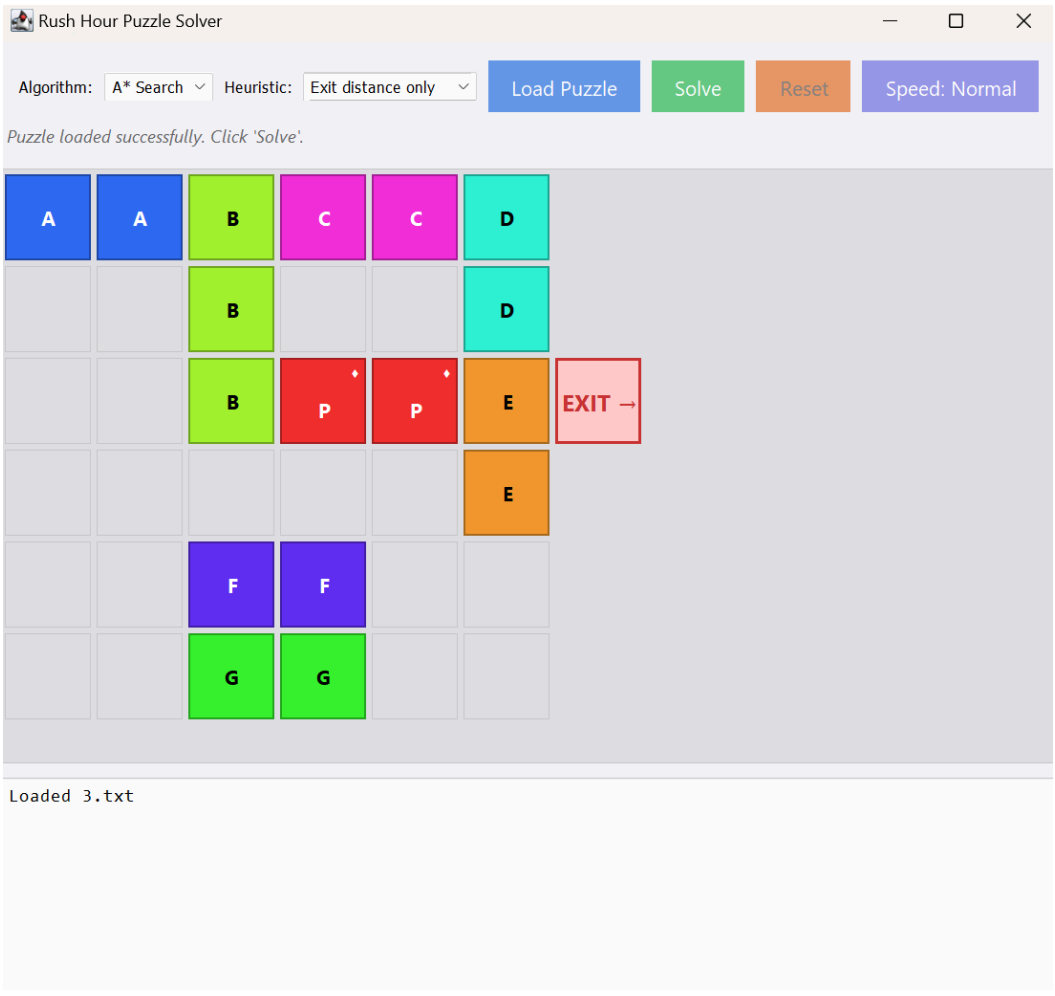

Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
Ready to solve or load a new puzzle
P
P
EXIT →
B
B
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.

Output

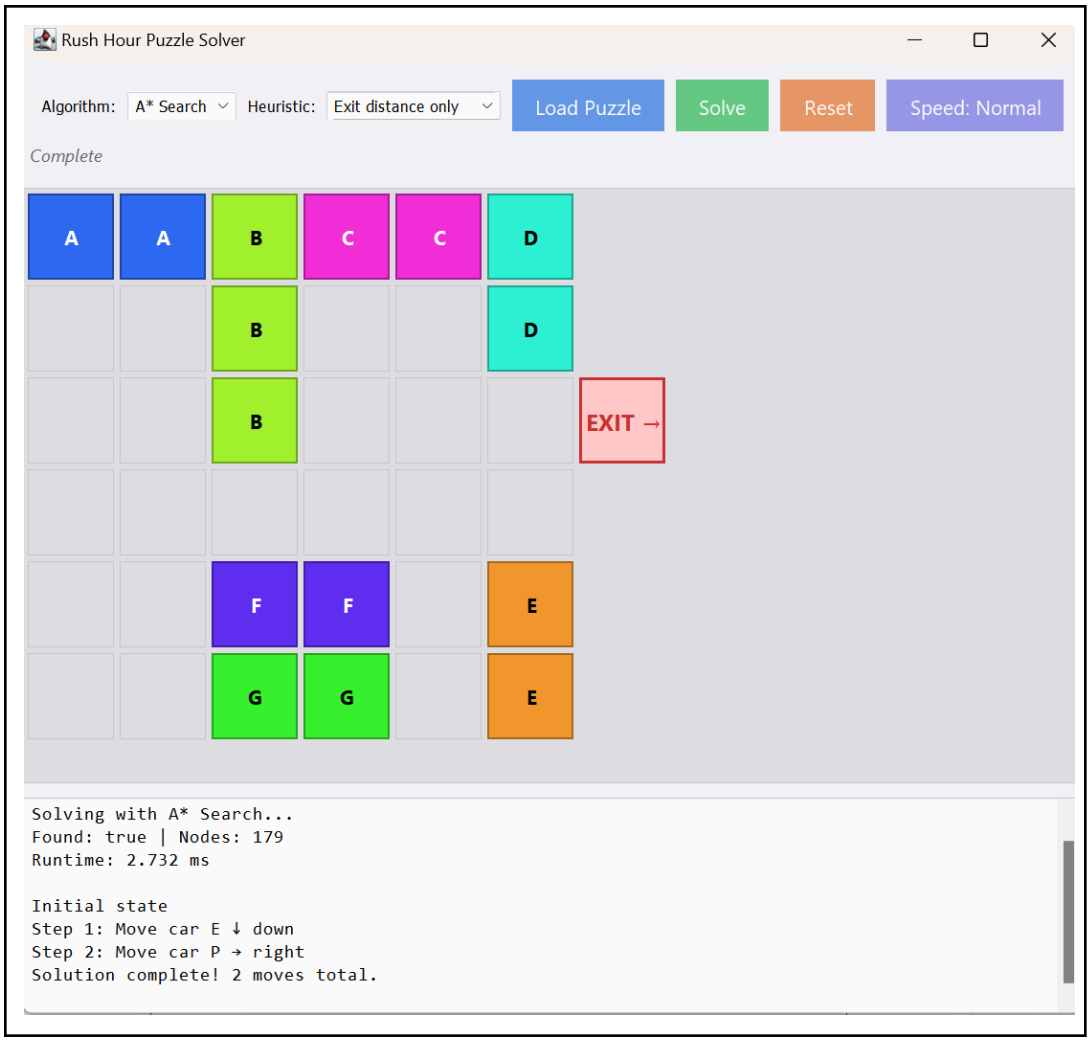


Test Case 2

Input.txt

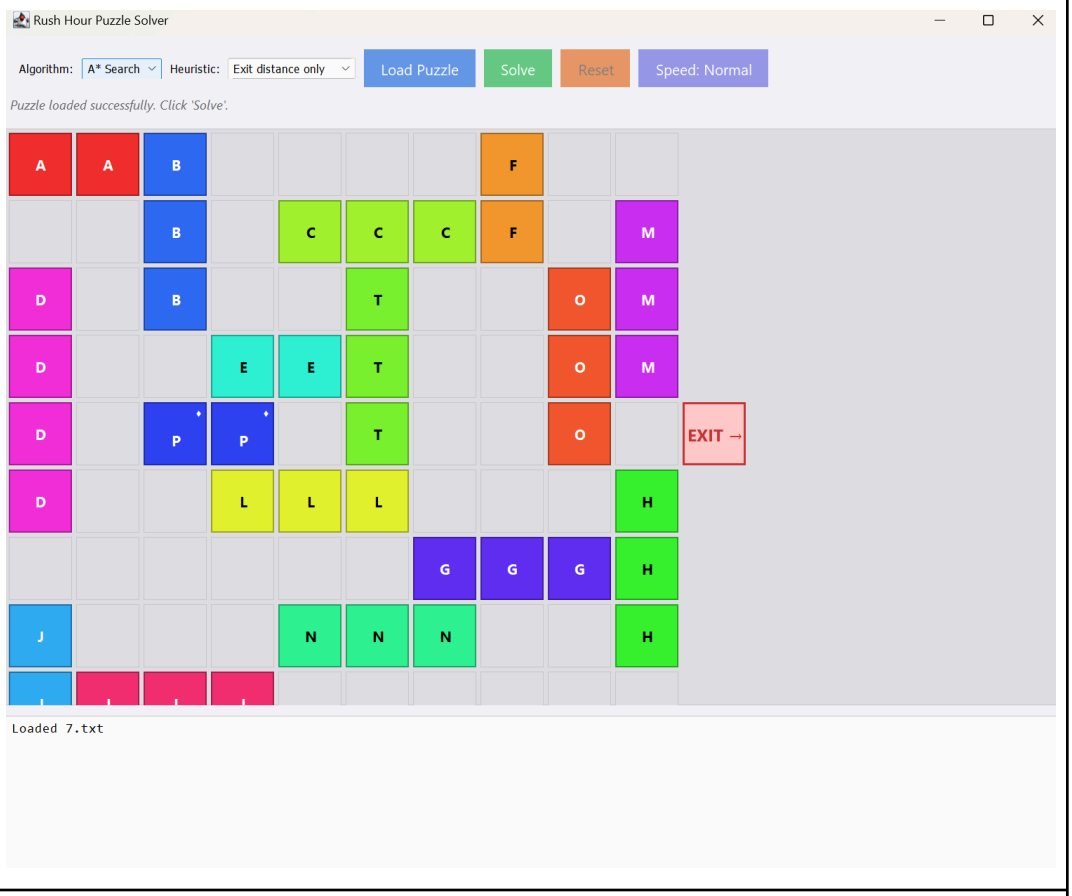

Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
A
A
B
C
C
D
B
D
B
P
P
E
EXIT →
E
F
F
G
G
Loaded 3.txt

Output

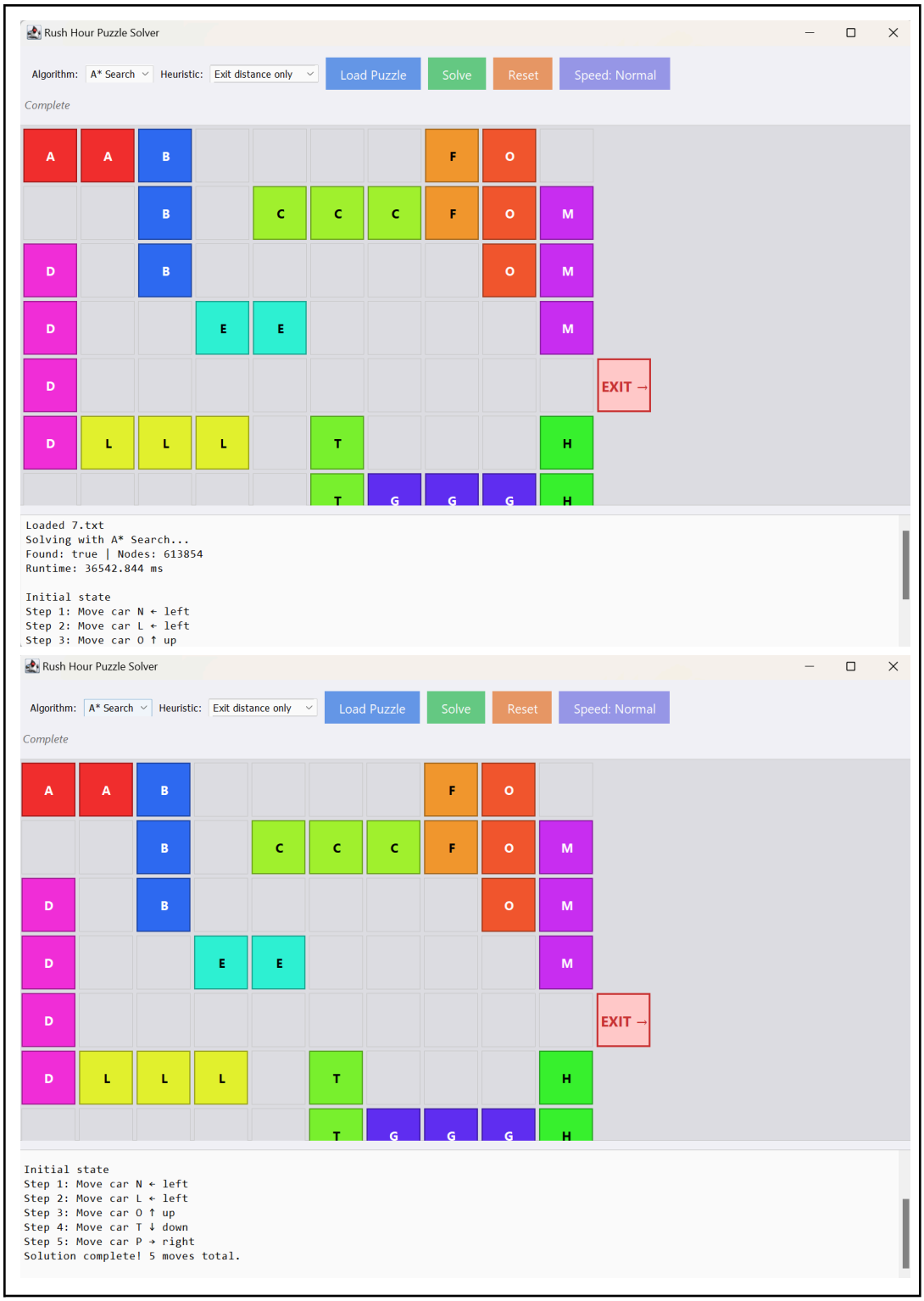


Test Case 3

Input.txt

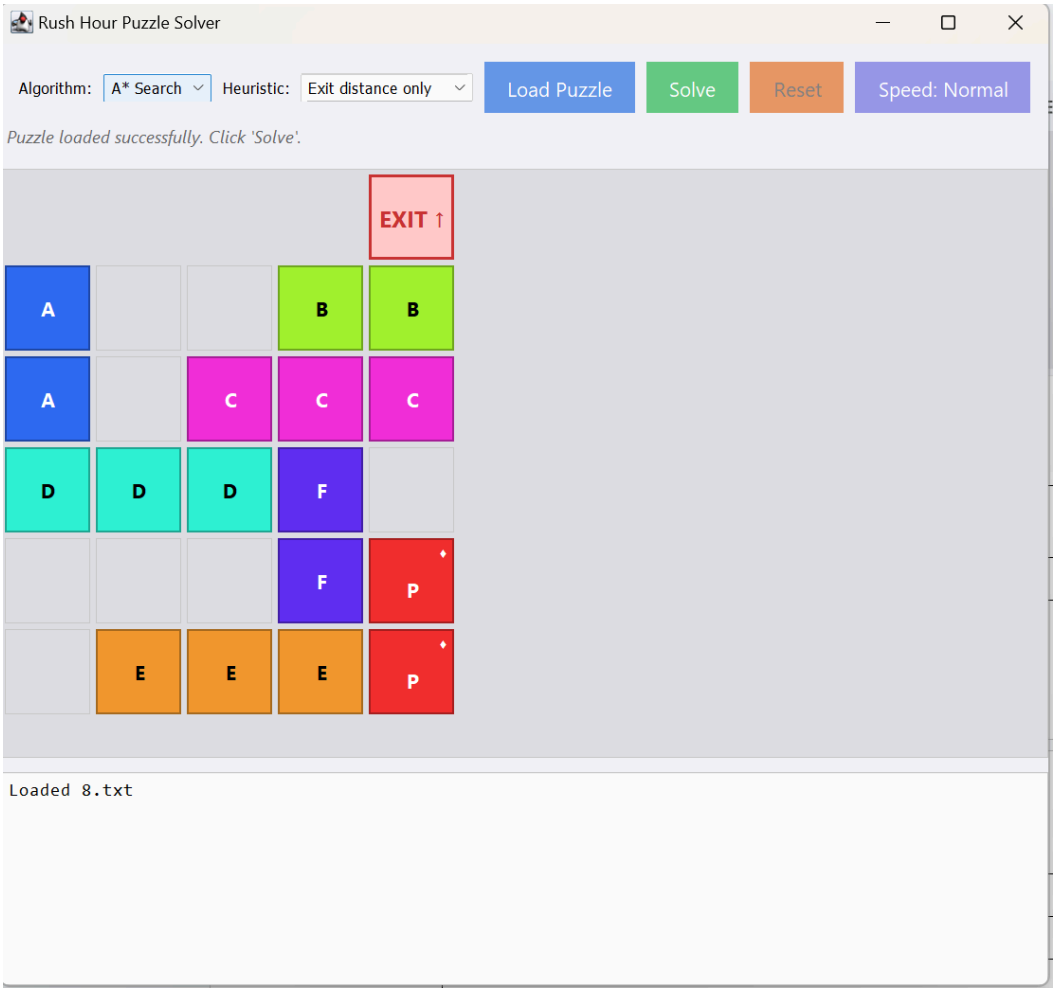

Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT →
Loaded 7.txt

Output

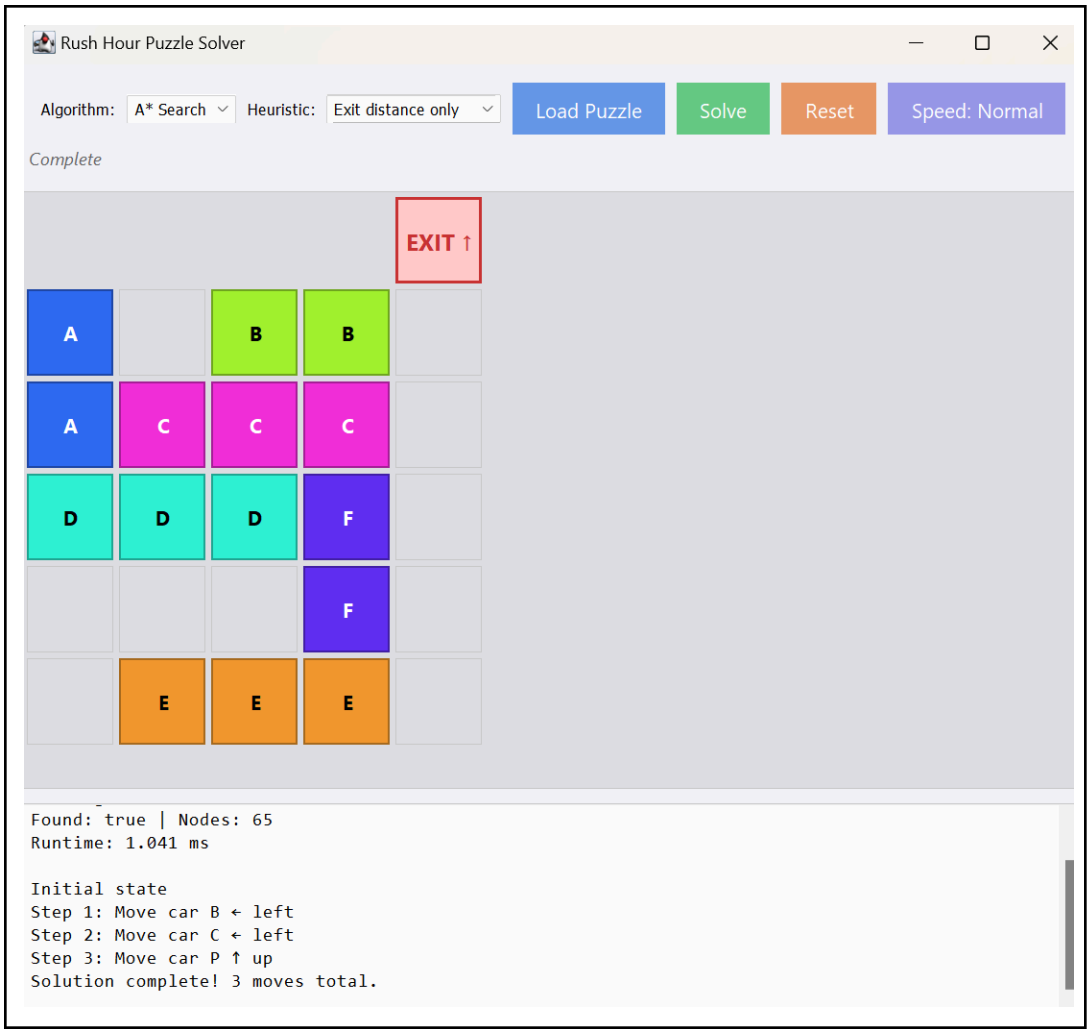


Test Case 4

Input.txt


Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Exit distance only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
EXIT ↑
A
B
B
A
C
C
C
D
D
D
F
F
P
E
E
E
P
Loaded 8.txt

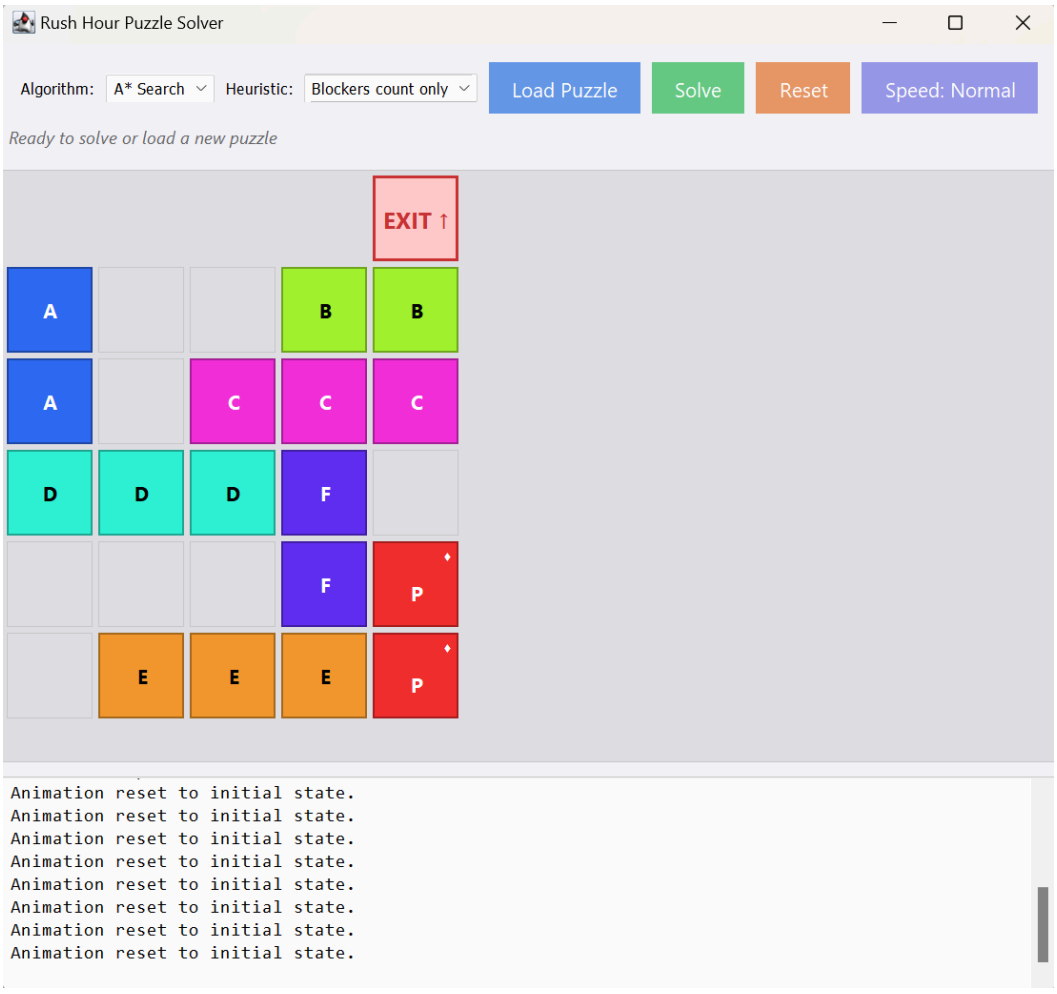
Output



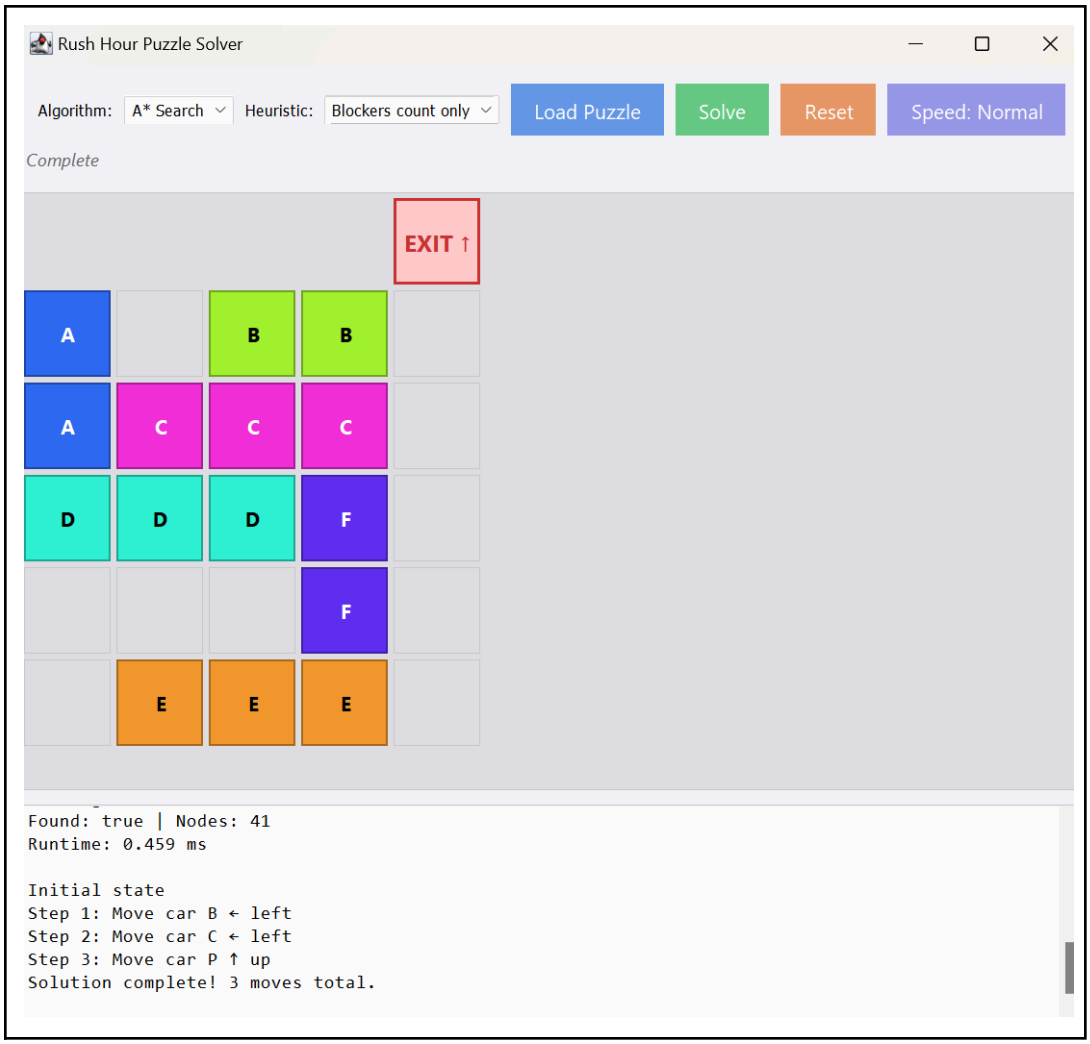
3.3.3.2 Blockers Count Heuristic

Test Case 1

Input.txt

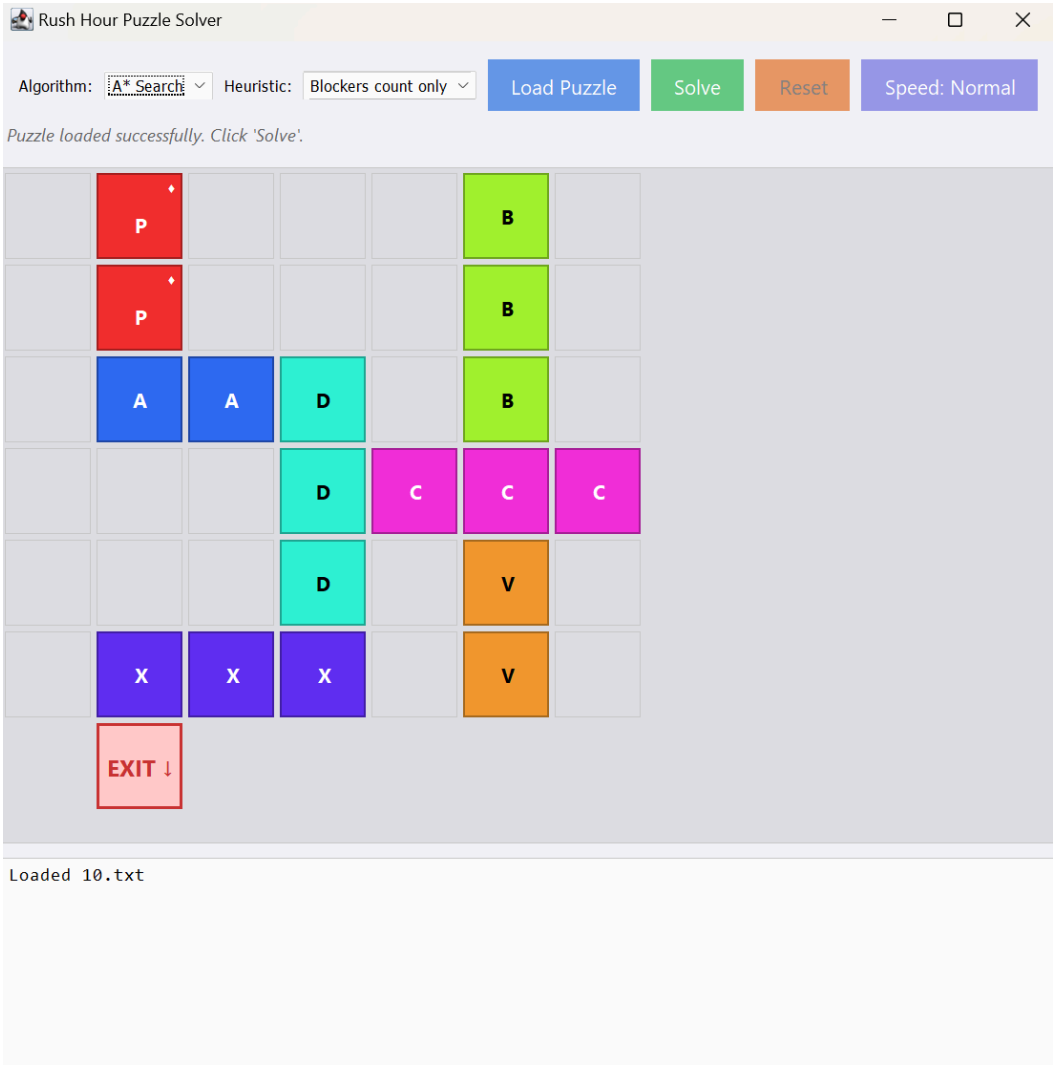

Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Blockers count only
Load Puzzle
Solve
Reset
Speed: Normal
Ready to solve or load a new puzzle
EXIT ↑
A
B
B
A
C
C
C
D
D
D
F
F
P
E
E
E
P
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.
Animation reset to initial state.

Output

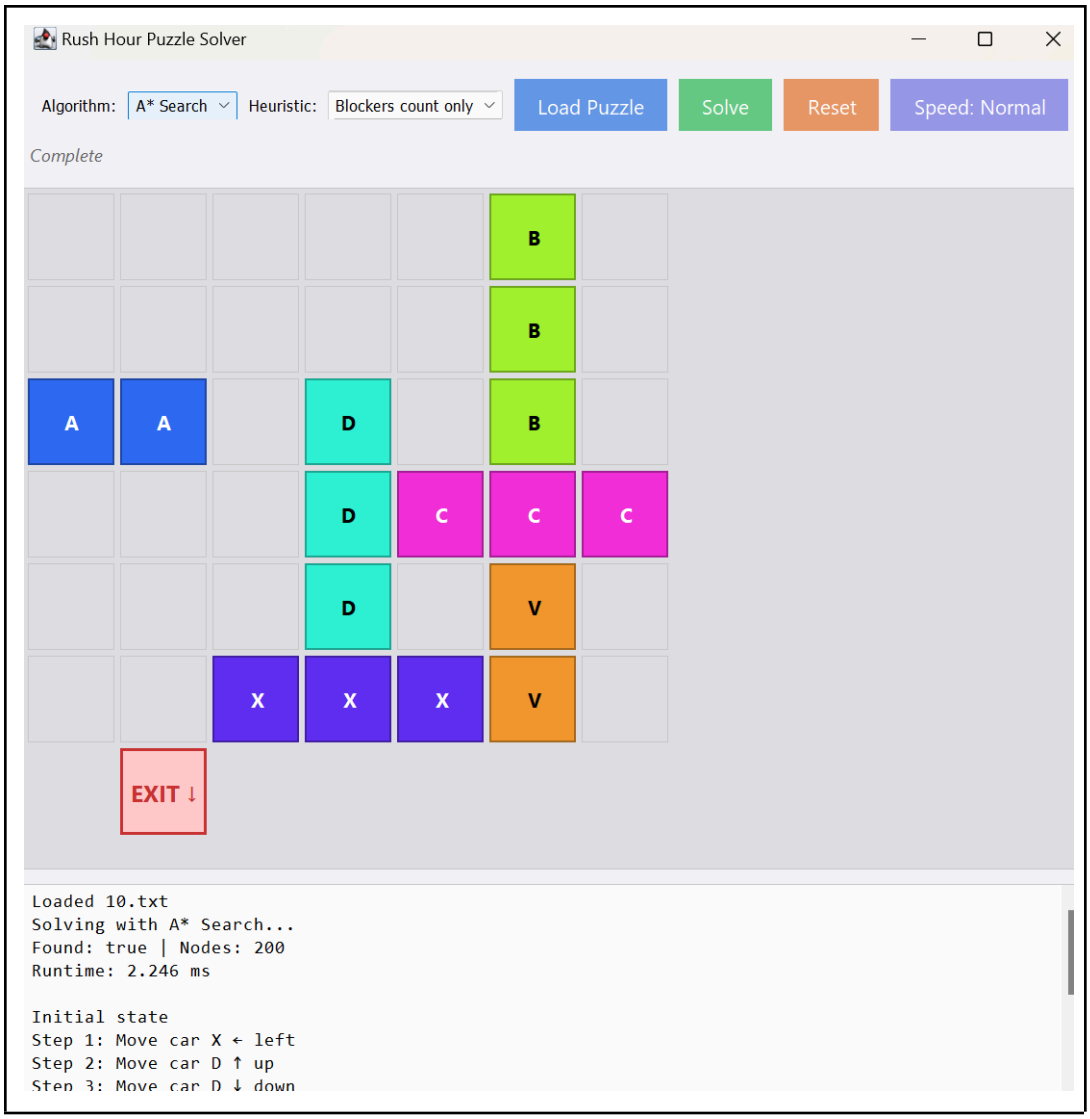


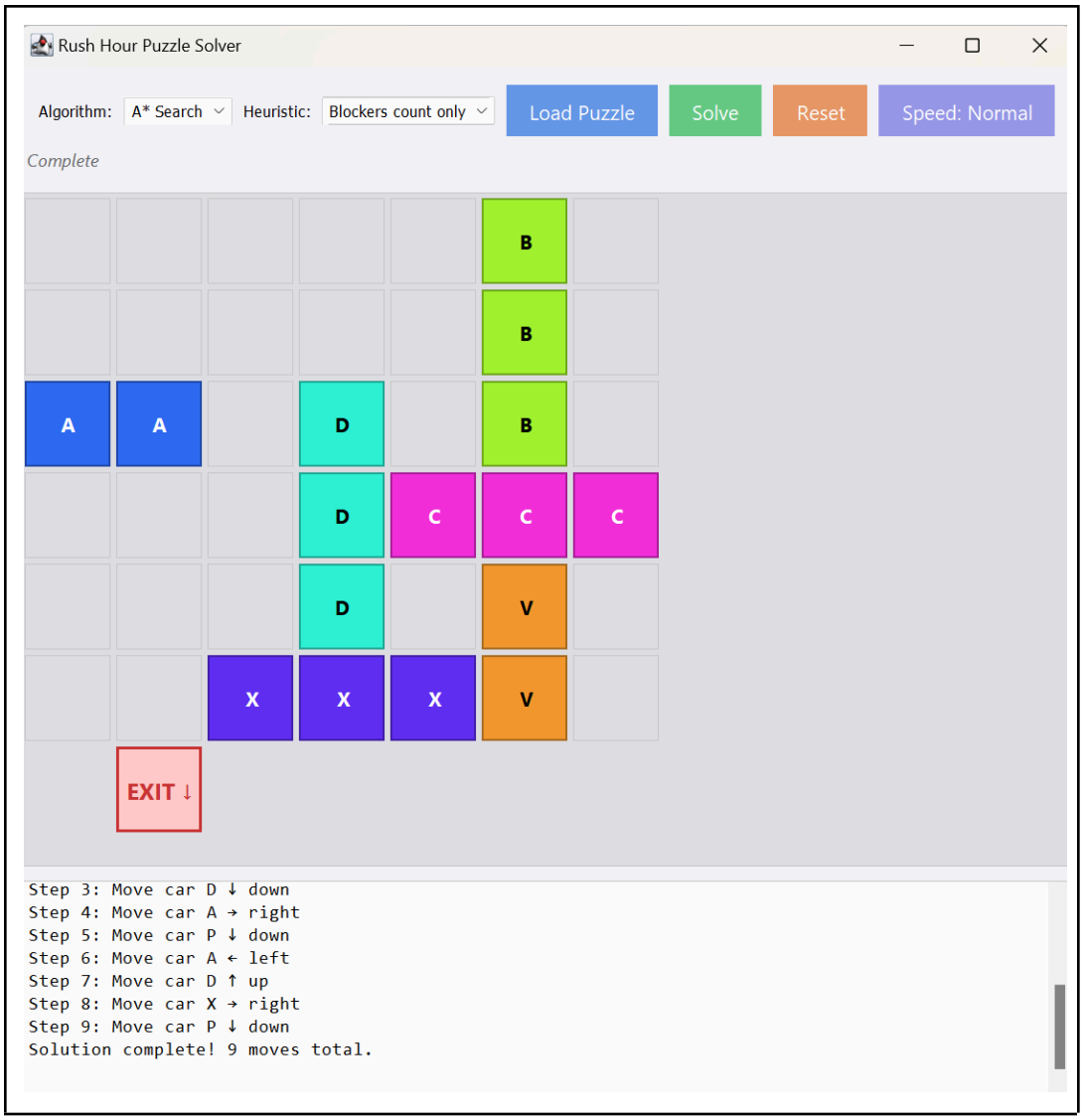
Test Case 2

Input.txt


Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Blockers count only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
P
P
B
B
A
A
D
B
D
C
C
C
D
V
X
X
X
V
EXIT ↓
Loaded 10.txt

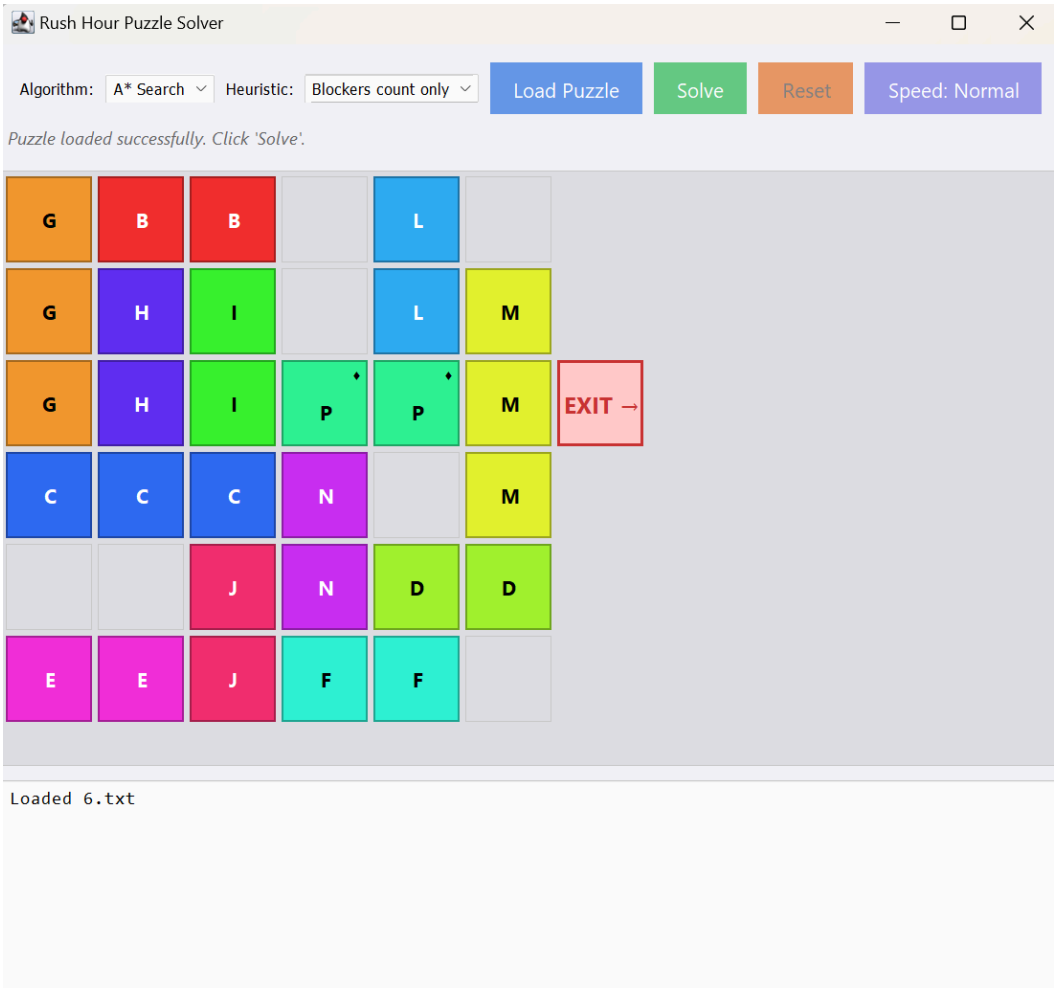
Output


Rush Hour Puzzle Solver
Algorithm: A* Search
Heuristic: Blockers count only
Load Puzzle
Solve
Reset
Speed: Normal
Complete
B
B
A
A
D
B
D
C
C
C
D
V
X
X
X
V
EXIT ↓
Loaded 10.txt
Solving with A* Search...
Found: true | Nodes: 200
Runtime: 2.246 ms
Initial state
Step 1: Move car X ← left
Step 2: Move car D ↑ up
Step 3: Move car D ↓ down

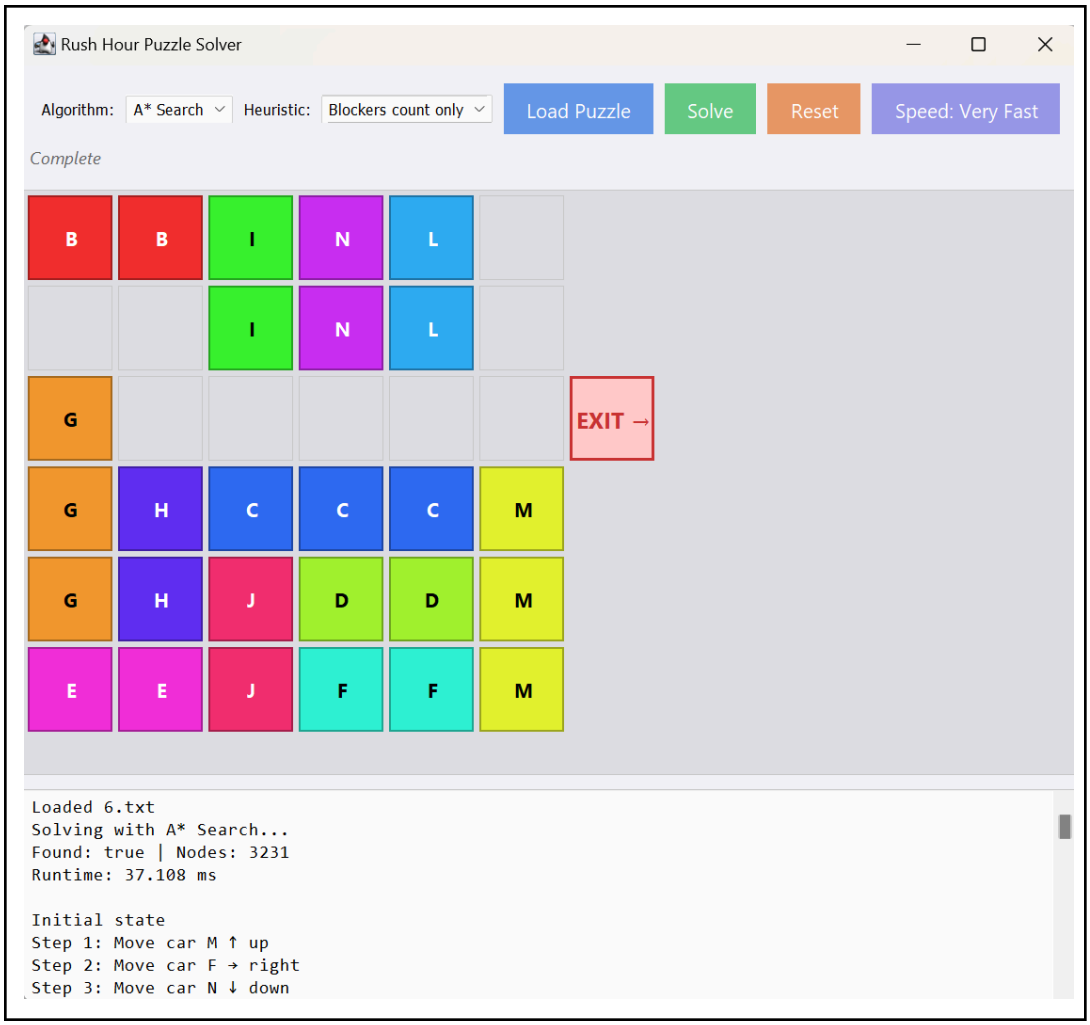


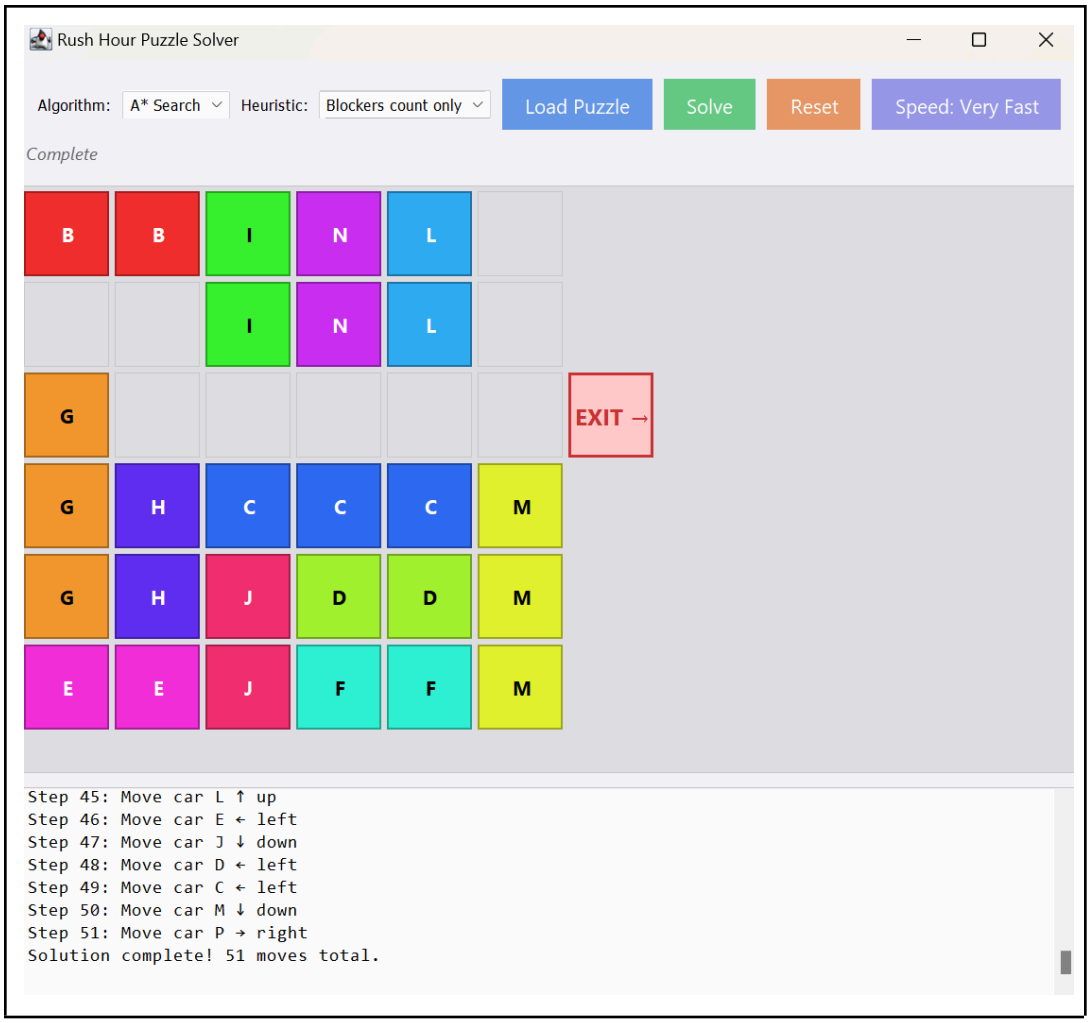
Test Case 3

Input.txt


Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Blockers count only
Load Puzzle
Solve
Reset
Speed: Normal
Puzzle loaded successfully. Click 'Solve'.
G B B L
G H I L M
G H I P P M EXIT →
C C C N M
J N D D
E E J F F
Loaded 6.txt

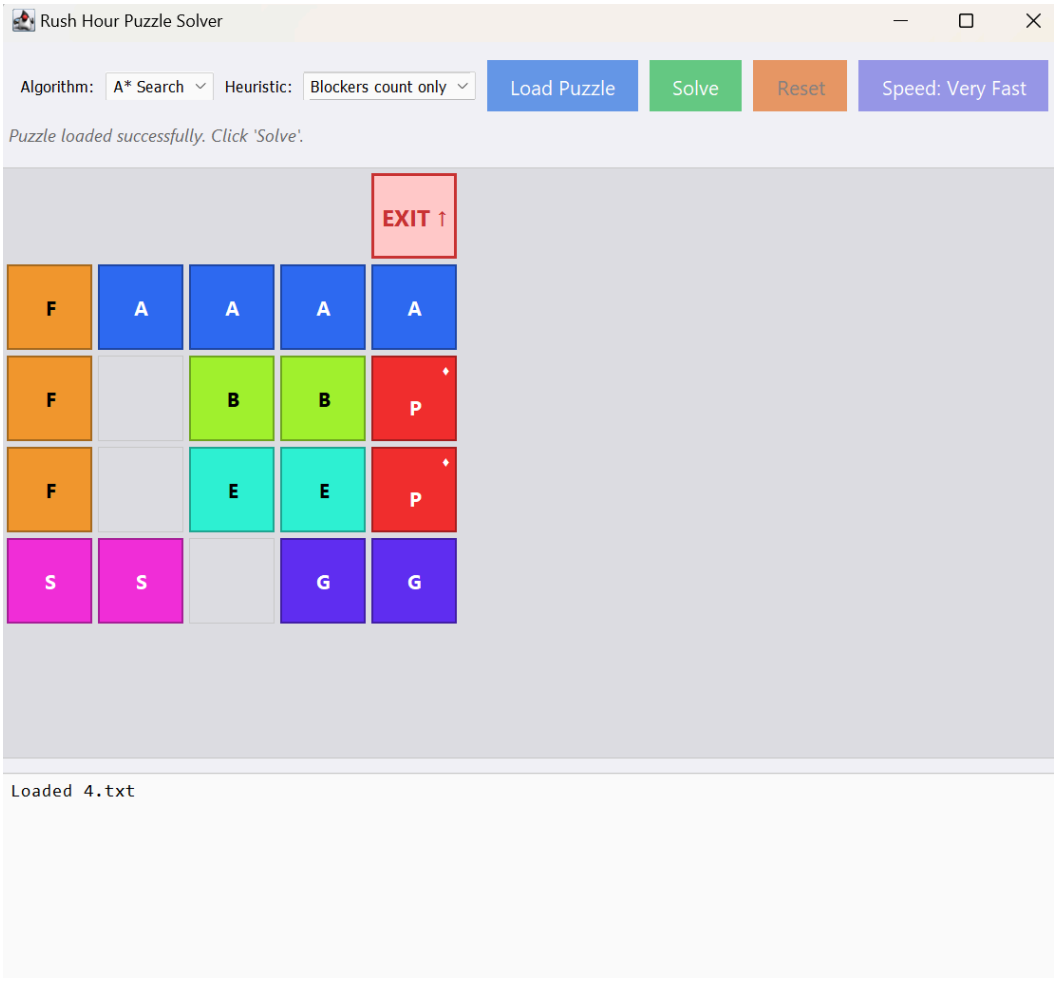
Output


Rush Hour Puzzle Solver
Algorithm: A* Search
Heuristic: Blockers count only
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
B B I N L
I N L
G
EXIT →
G H C C C M
G H J D D M
E E J F F M
Loaded 6.txt
Solving with A* Search...
Found: true | Nodes: 3231
Runtime: 37.108 ms
Initial state
Step 1: Move car M ↑ up
Step 2: Move car F → right
Step 3: Move car N ↓ down

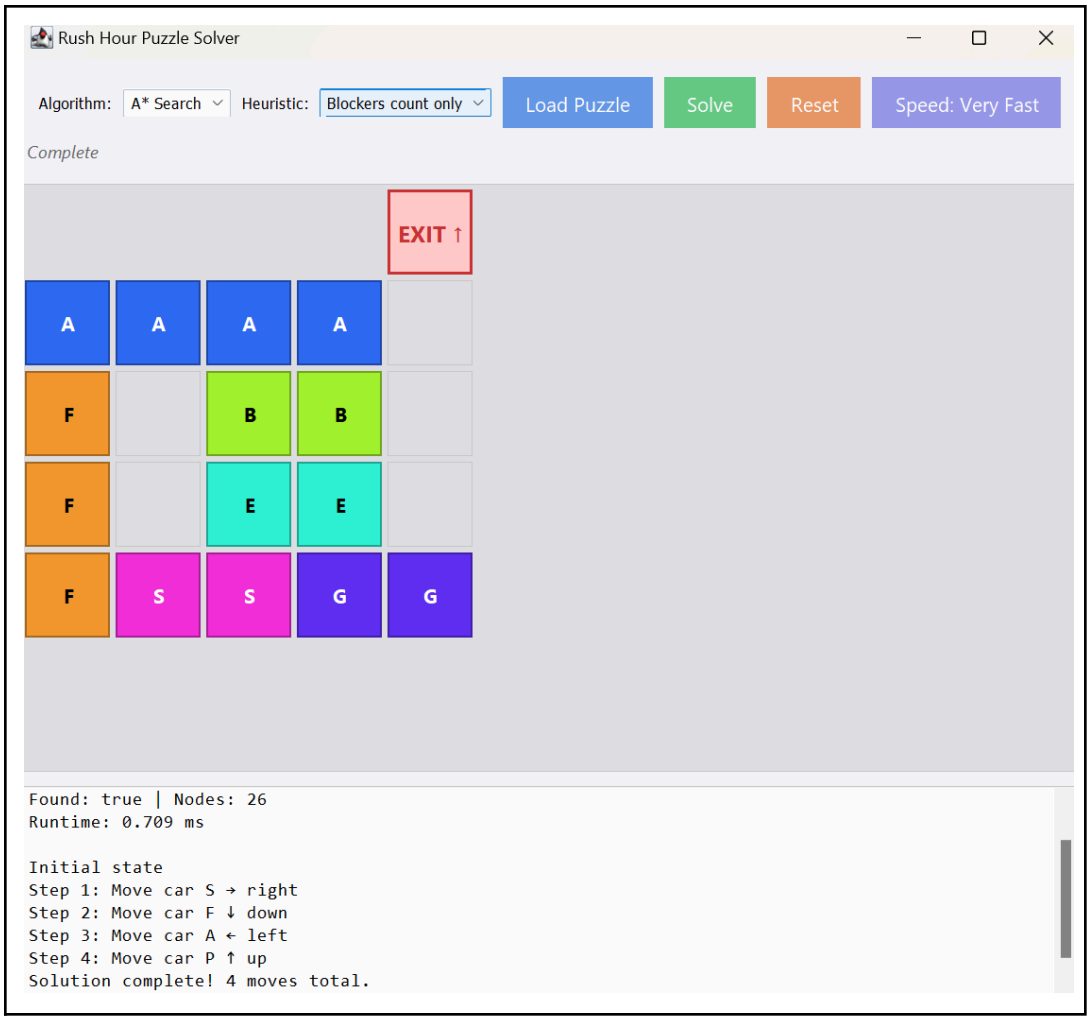


Test Case 4

Input.txt


Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Blockers count only
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
EXIT
F
A
A
A
A
F
B
B
P
F
E
E
P
S
S
G
G
Loaded 4.txt

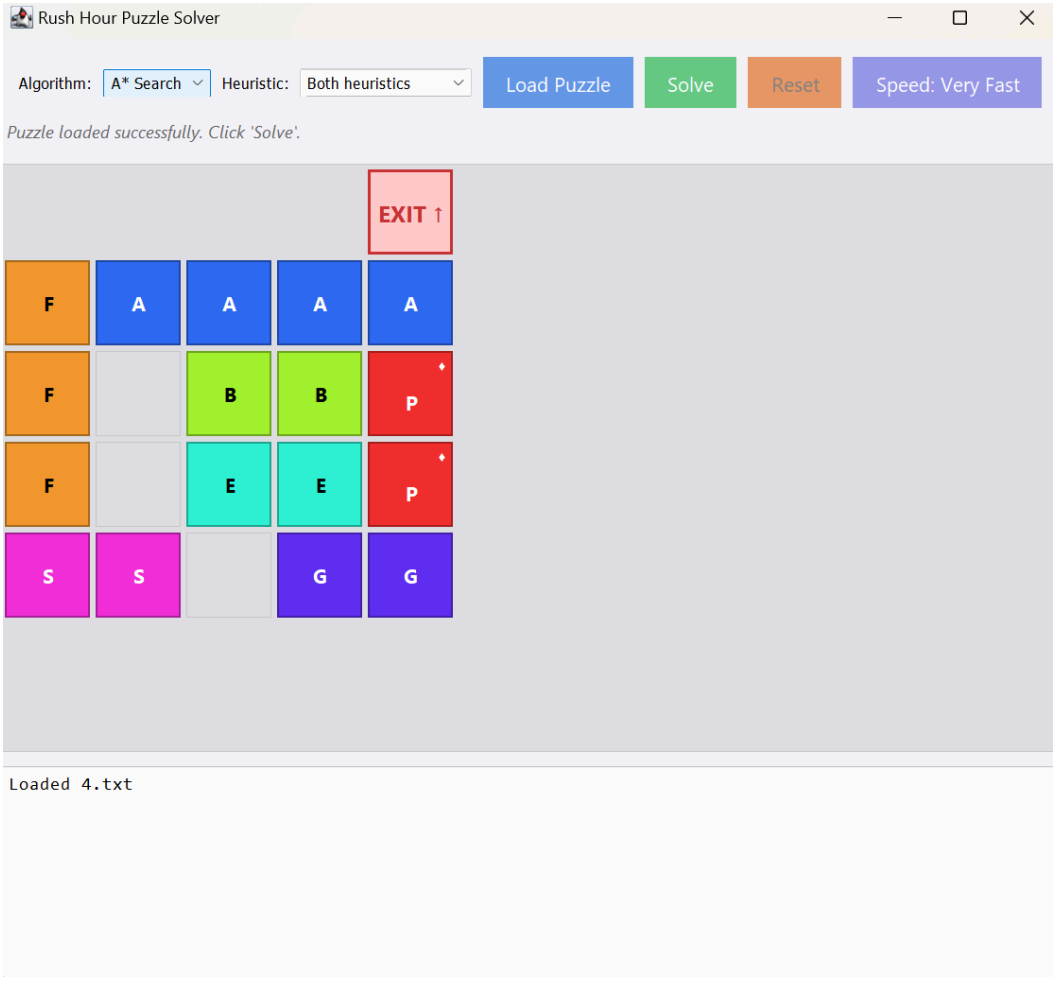
Output



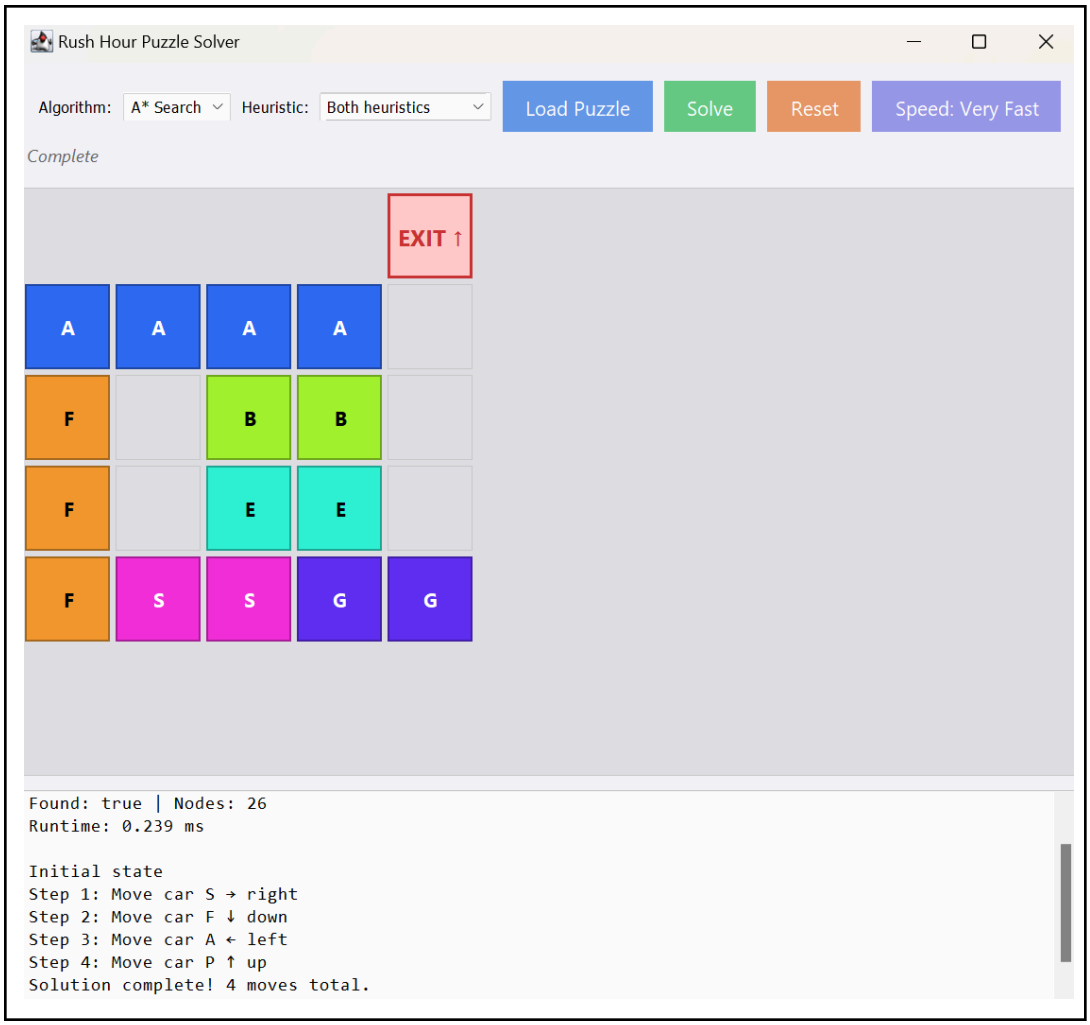
3.3.3.3 Combined Heuristic (Exit Distance & Blockers Count)

Test Case 1

Input.txt

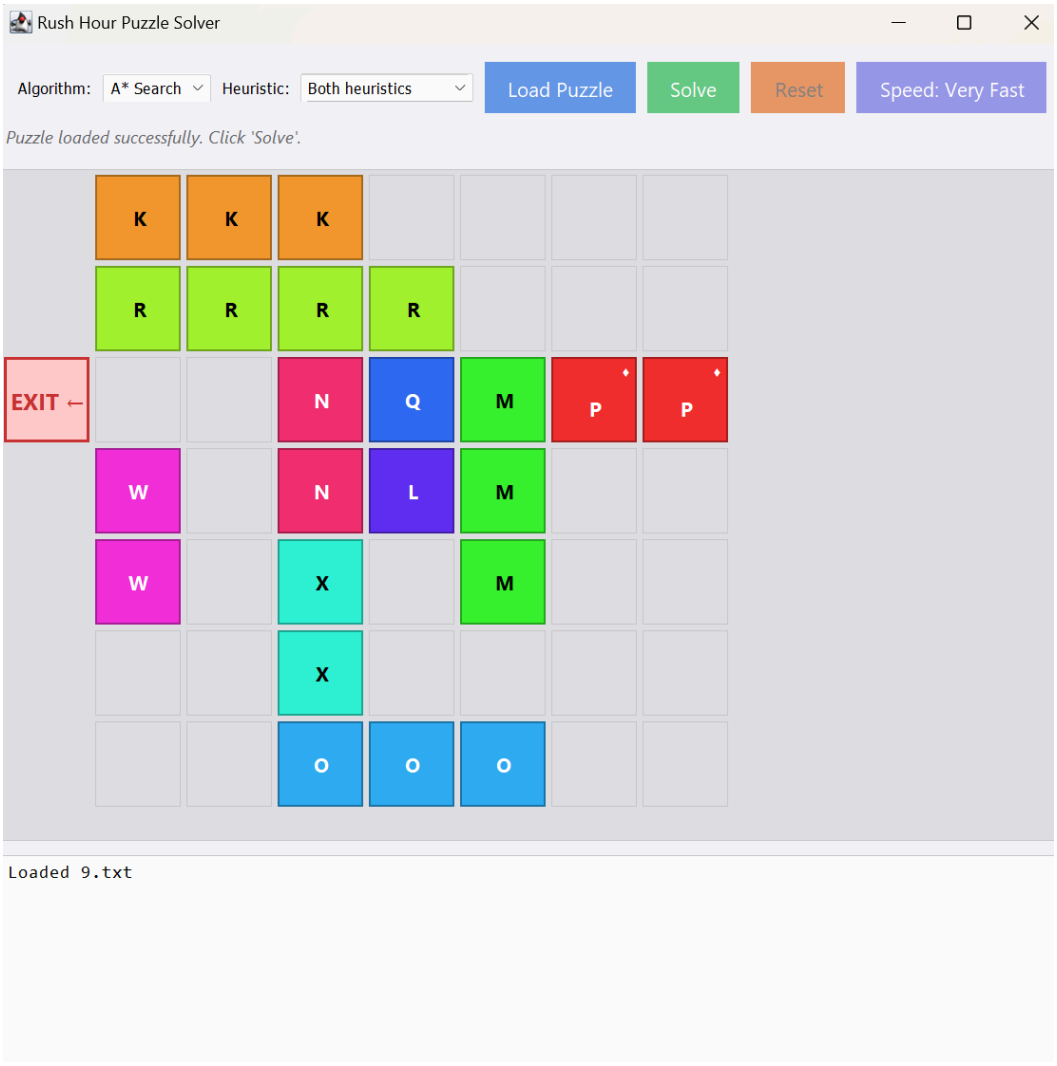

Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Both heuristics
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
EXIT ↑
F A A A A
F B B P
F E E P
S S G G
Loaded 4.txt

Output

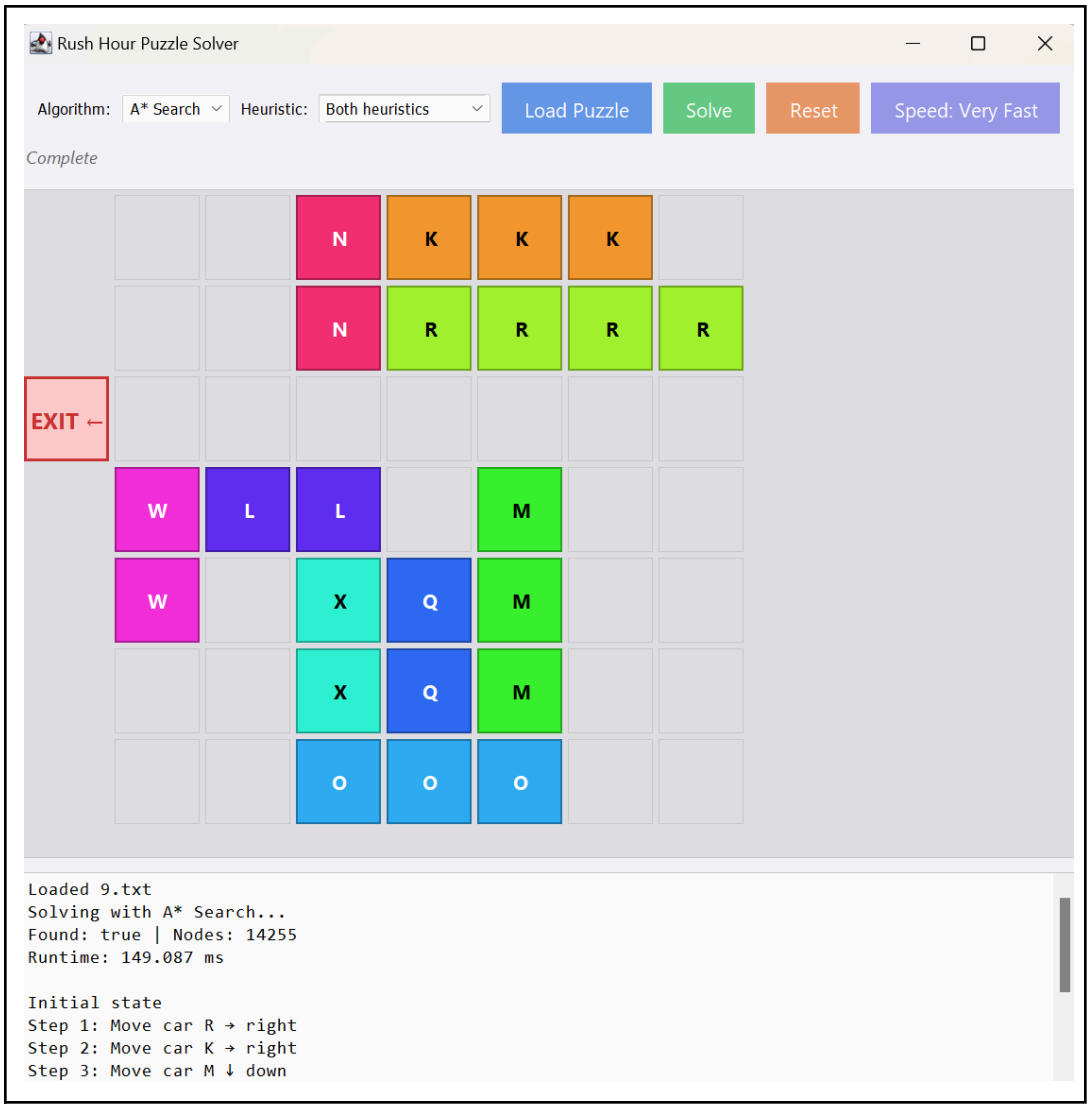


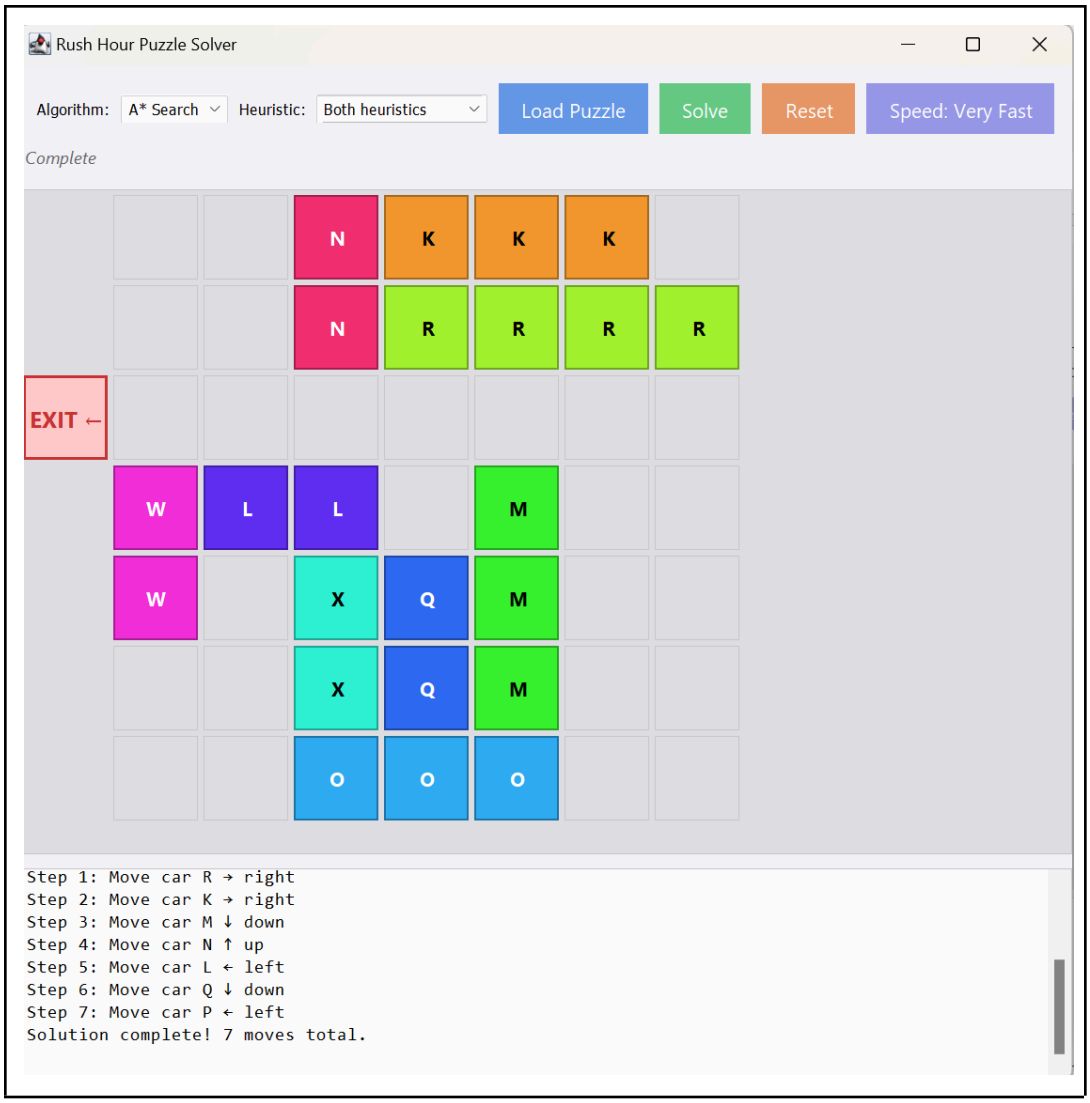
Test Case 2

Input.txt


Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Both heuristics
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
EXIT ←
Loaded 9.txt

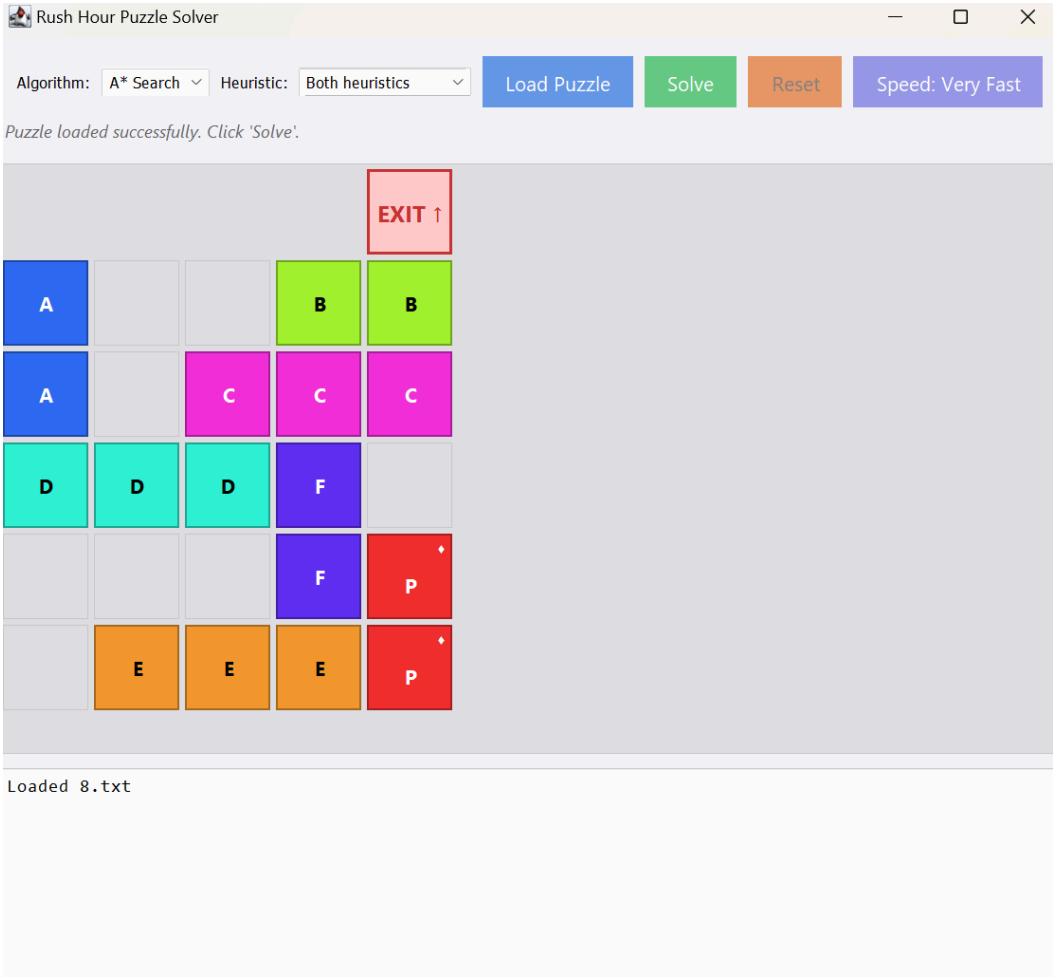
Output


Rush Hour Puzzle Solver
Algorithm: A* Search
Heuristic: Both heuristics
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
EXIT ←
Loaded 9.txt
Solving with A* Search...
Found: true | Nodes: 14255
Runtime: 149.087 ms
Initial state
Step 1: Move car R → right
Step 2: Move car K → right
Step 3: Move car M ↓ down

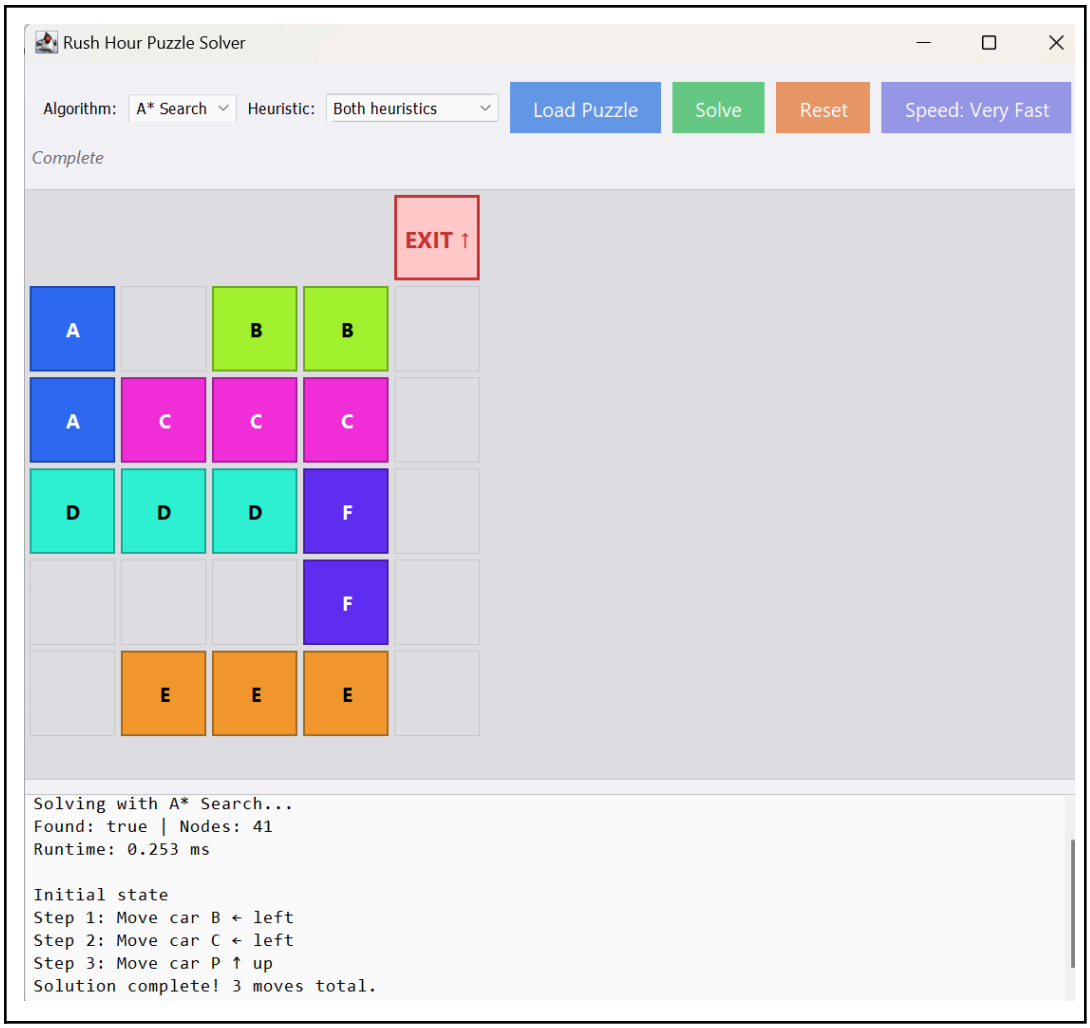


Test Case 3

Input.txt

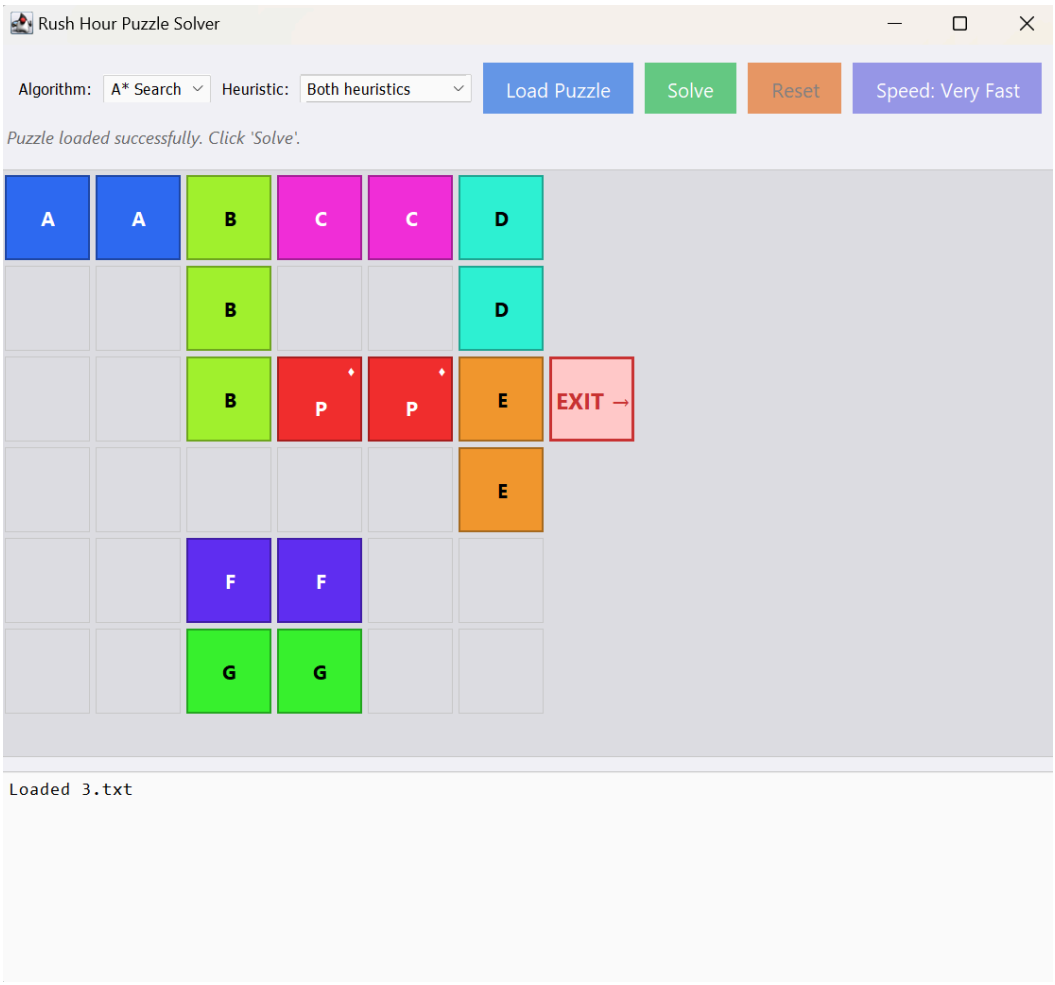

Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Both heuristics
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
EXIT ↑
A
B
B
A
C
C
C
D
D
D
F
F
P
E
E
E
P
Loaded 8.txt

Output

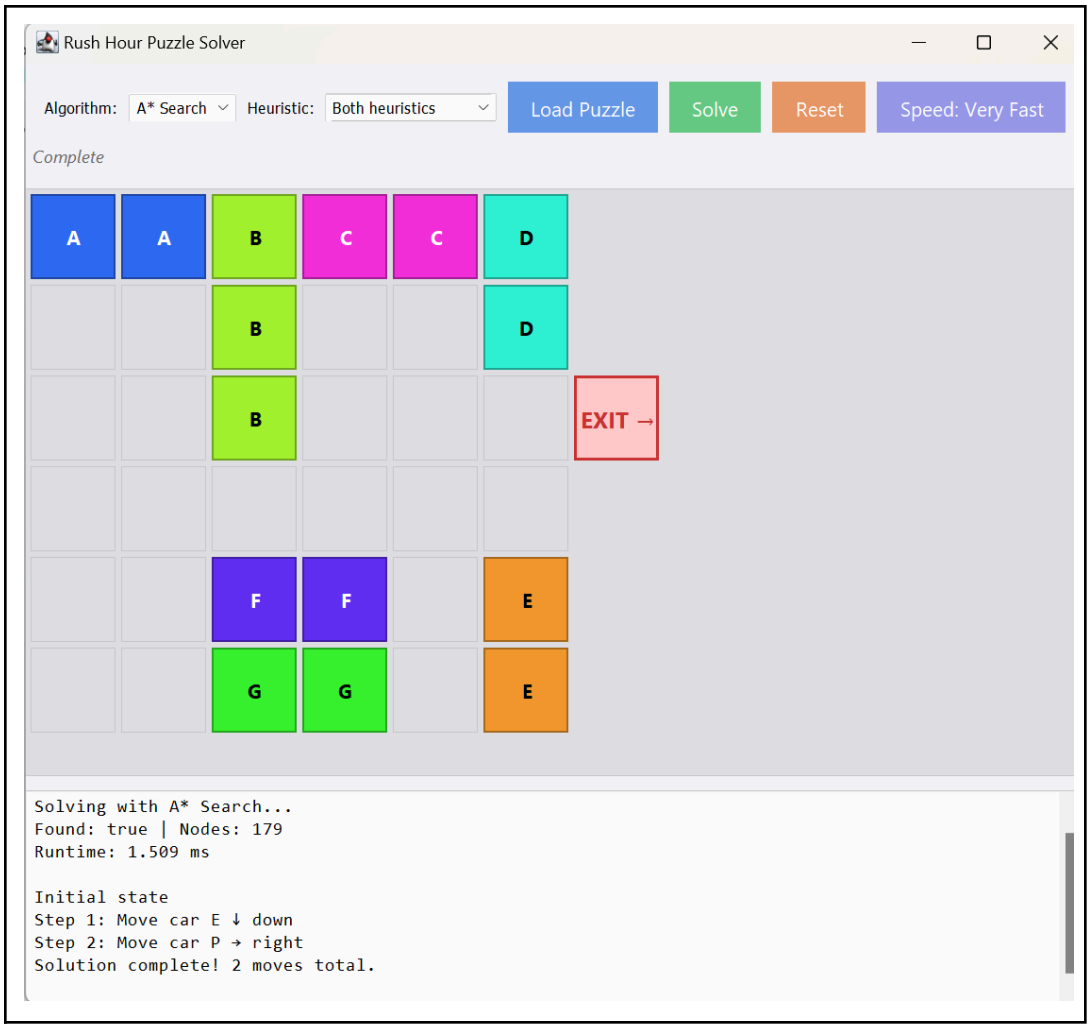


Test Case 4

Input.txt


Rush Hour Puzzle Solver
Algorithm:
A* Search
Heuristic:
Both heuristics
Load Puzzle
Solve
Reset
Speed: Very Fast
Puzzle loaded successfully. Click 'Solve'.
A
A
B
C
C
D
B
D
B
P
P
E
EXIT →
E
F
F
G
G
Loaded 3.txt

Output


Rush Hour Puzzle Solver
Algorithm: A* Search
Heuristic: Both heuristics
Load Puzzle
Solve
Reset
Speed: Very Fast
Complete
A A B C C D
B D
B EXIT →
F F E
G G E
Solving with A* Search...
Found: true | Nodes: 179
Runtime: 1.509 ms
Initial state
Step 1: Move car E ↓ down
Step 2: Move car P → right
Solution complete! 2 moves total.

3.4. Analisis Hasil Pengujian

Dalam implementasi proyek ini, tiga algoritma pencarian jalur digunakan untuk menyelesaikan permasalahan Rush Hour Puzzle, yaitu Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A*. Setiap algoritma diuji terhadap berbagai konfigurasi papan dengan ukuran dan kompleksitas berbeda untuk mengamati performa serta efisiensi pencarian solusi.

Kompleksitas dan Efisiensi:

1. Uniform Cost Search (UCS):
    - Kompleksitas waktu: O(b^d), di mana b adalah branching factor dan d adalah kedalaman solusi terpendek.
    - UCS menjamin solusi optimal karena selalu memilih simpul dengan cost terendah (g(n)) terlebih dahulu.
    - Namun, karena tidak menggunakan heuristik, UCS cenderung mengeksplorasi banyak node yang tidak mengarah ke solusi, terutama ketika jalur yang benar berada jauh dari akar atau dikelilingi banyak kemungkinan.
2. Greedy Best First Search (GBFS):
    - Kompleksitas waktu: secara teoritis bisa mendekati O(b^m), di mana m adalah maksimum kedalaman search tree.
    - GBFS hanya mempertimbangkan nilai heuristik h(n) tanpa memperhatikan cost aktual dari simpul awal.
    - Meskipun GBFS dapat menemukan solusi lebih cepat pada kasus sederhana, tidak menjamin solusi optimal, dan sangat bergantung pada kualitas heuristik yang digunakan.
    - Pada beberapa kasus kompleks, GBFS dapat "terjebak" dalam jalur yang tampak menjanjikan secara heuristik namun sebenarnya salah arah.
3. A* (A Star):
    - Kompleksitas waktu: O(b^d) dalam kasus terbaik jika heuristik admissible dan consistent.
    - A* menggabungkan cost aktual (g(n)) dan estimasi ke tujuan (h(n)) melalui fungsi f(n) = g(n) + h(n), yang membuatnya mampu menemukan solusi optimal secara efisien.

- Dalam pengujian, A* menunjukkan performa paling seimbang antara optimalitas dan kecepatan, terutama saat menggunakan kombinasi heuristik (jarak ke pintu keluar + jumlah penghalang).

berdasarkan eksperimen:

- Pada papan kecil, semua algoritma dapat menyelesaikan masalah dengan waktu relatif singkat, walaupun GBFS tetap memiliki keunggulan dalam segi runtime, yaitu paling singkat.
- Pada papan menengah hingga besar:
  - GBFS paling cepat menemukan solusi (dalam milidetik) tapi kadang solusinya kurang optimal dimana GBFS memiliki langkah yang jauh lebih panjang dibandingkan dengan solusi lainnya.
  - UCS menjamin solusi optimal, tetapi membutuhkan waktu dan eksplorasi node yang jauh lebih banyak.
  - A* memiliki performa terbaik secara keseluruhan, dengan keseimbangan waktu dan kualitas solusi, A* memiliki waktu yang lebih cepat dibanding UCS dan dengan jumlah langkah yang lebih sedikit dibanding GBFS.

3.5. Implementasi Bonus

3.5.1. Penggunaan GUI

Program ini menggunakan antarmuka grafis berbasis Java Swing yang dirancang untuk mempermudah pengguna dalam memuat, memilih algoritma, dan memvisualisasikan solusi puzzle Rush Hour. Pengguna dapat memilih algoritma UCS, GBFS, atau A* serta menyesuaikan heuristik untuk GBFS dan A*. GUI ini memungkinkan pengguna memuat file puzzle melalui tombol “Load Puzzle”, menjalankan pencarian solusi dengan “Solve”, dan mengatur kecepatan animasi menggunakan tombol “Speed”. Setiap langkah solusi divisualisasikan dengan animasi di papan permainan dengan warna unik untuk setiap mobil. Selain itu, GUI juga dilengkapi log area dan status bar untuk memberikan informasi proses solving secara real-time.

3.5.2 *Multiple Heuristics*

Khusus untuk algoritma Greedy Best-First Search dan A* search, program menyediakan (untuk masing-masing algoritma) 3 opsi heuristik yang dapat dipilih untuk pencarian solusi. Heuristik pertama adalah menghitung jarak antara mobil primer dengan pintu keluar. Nilai ini, sesuai namanya, menghitung jumlah sel yang berada di antara mobil primer dan pintu keluar untuk digunakan sebagai nilai heuristik. Semakin dekat mobil primer terhadap pintu keluar, maka nilai heuristik algoritma semakin kecil. Heuristik kedua adalah menghitung jumlah mobil yang menghalangi jalur mobil primer menuju pintu keluar. Penggunaan nilai heuristik ini serupa dengan penggunaan nilai heuristik pertama. Apabila semakin sedikit mobil yang menghalangi jalur keluar, maka semakin kecil nilai heuristik pada algoritma. Terakhir, heuristik yang digunakan adalah gabungan dari kedua algoritmanya. Nilai heuristik ini merupakan jumlah dari nilai heuristik pertama (jarak ke pintu keluar) dan heuristik kedua (jumlah mobil penghalang). Ketiga heuristik tersebut diimplementasikan pada algoritma GBFS dan juga A*.

BAB IV
LAMPIRAN

Pranala repository: https://github.com/faawibowo/Tucil3_13523153_13523140

Poin		
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	✔	