

**IF2211 STRATEGI ALGORITMA**  
**TUGAS KECIL 3**

**Penyelesaian Puzzle Rush Hour**  
**Menggunakan Algoritma Pathfinding**



Diusun Oleh:

Bryan Ho 13523029

Alvin Christopher Santausa 13523033

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**JL. GANESA 10, BANDUNG 40132**  
**2025**

## DAFTAR ISI

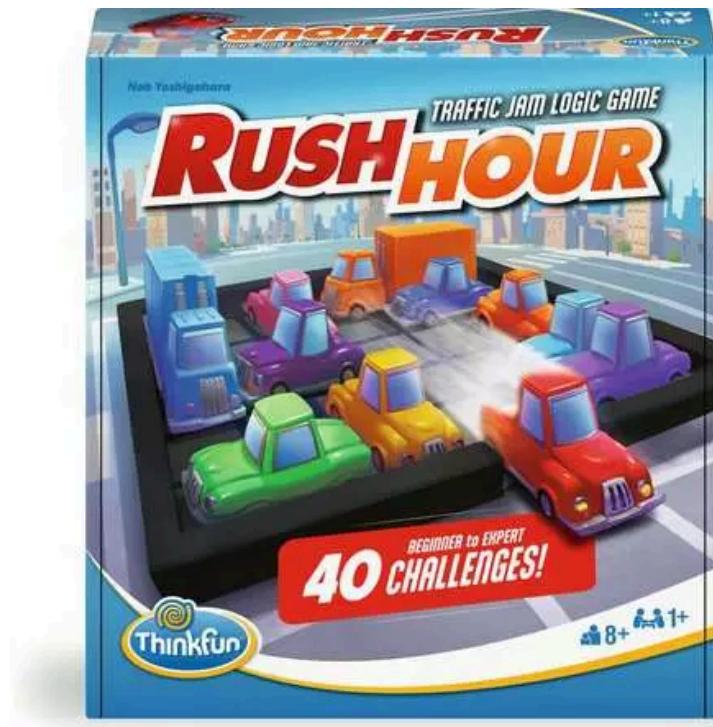
<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I</b>	<b>4</b>
<b>LATAR BELAKANG MASALAH</b>	<b>4</b>
<b>BAB II</b>	<b>6</b>
<b>DASAR TEORI</b>	<b>6</b>
2.1 Algoritma Uniform Cost Search (UCS)	6
2.2 Algoritma Greedy Best First Search (GBFS)	8
2.3 Algoritma A*	9
<b>BAB III</b>	<b>14</b>
<b>ANALISIS ALGORITMA</b>	<b>14</b>
3.1 Fungsi Evaluasi $f(n)$ dan $g(n)$	14
3.2 Keadmissibilitasan Heuristik pada A*	14
3.3 Perbandingan UCS dan BFS pada Penyelesaian Rush Hour	15
3.4 Efisiensi Algoritma A* dibandingkan UCS pada Penyelesaian Rush Hour	15
3.5 Optimalitas Greedy Best First Search dalam Rush Hour	15
<b>BAB IV</b>	<b>17</b>
<b>IMPLEMENTASI</b>	<b>17</b>
4.1 Algoritma	17
4.2 Heuristik	21
4.3 Model	23
4.4 I/O	26
4.5 Komponen	28
4.6 Pages	32
<b>BAB V</b>	<b>35</b>
<b>PENGUJIAN DAN HASIL ANALISIS</b>	<b>35</b>
5.1 Hasil Pengujian	35
a. Test case 1	35
b. Test case 2	37
c. Test case 3	40
d. Test case 4	42
e. Test case 5	45
f. Test case 6	48
g. Test case 7	48
h. Test case 8	49

5.2 Analisis Pengujian	49
5.2.1 Algoritma Uniform Cost Search (UCS)	49
5.2.2 Algoritma Greedy Best First Search (GBFS)	49
5.2.3 Algoritma A*	49
5.2.4 Algoritma Fringe Search	50
<b>BAB VI</b>	<b>51</b>
<b>PENJELASAN BONUS</b>	<b>51</b>
6.1 Algoritma Fringe Search	51
6.2 Fungsi Heuristik	51
6.2.1 Manhattan Distance Heuristic	51
6.2.2 Blocking Vehicles Heuristic	51
6.2.3 Blocking Cells Heuristic	52
6.3 Graphical User Interface	52
<b>BAB VII</b>	<b>53</b>
<b>LAMPIRAN</b>	<b>53</b>
7.1 Daftar Pustaka	53
7.2 Tautan Repository	53
7.3 Tautan Deployment	53
7.4 Tabel Evaluasi	53

## BAB I

### LATAR BELAKANG MASALAH

**Deskripsi:**



**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.

## BAB II

### DASAR TEORI

#### 2.1 Algoritma Uniform Cost Search (UCS)

Uniform-Cost Search atau UCS adalah sebuah algoritma pencarian yang digunakan untuk menemukan jalur dengan total biaya minimum dalam graf berbobot. Termasuk ke dalam golongan uninformed search, UCS tidak memanfaatkan informasi tambahan tentang lokasi tujuan, melainkan hanya fokus pada biaya yang telah dikeluarkan sejauh ini dari titik awal.

Berbeda dengan algoritma Breadth-First Search (BFS) yang menganggap semua langkah memiliki bobot yang sama, UCS mempertimbangkan bobot atau biaya dari setiap lintasan. Ini menjadikannya lebih andal untuk skenario yang memerlukan efisiensi biaya, seperti mencari rute tercepat, penggunaan energi terhemat, atau solusi dengan waktu minimum.

UCS memanfaatkan priority queue sebagai struktur data utama, di mana setiap simpul (node) disimpan bersama total biaya dari simpul awal hingga simpul tersebut. Node dengan biaya terendah selalu diproses lebih dulu. Ketika sebuah node diambil dari antrian, algoritma akan mengecek apakah itu adalah solusi akhir. Jika ya, maka jalur paling efisien telah ditemukan. Jika belum, semua tetangga dari node tersebut dieksplorasi dan dimasukkan ke antrian, tentu saja dengan biaya kumulatif yang diperbarui.

Jika sebuah node ditemukan melalui jalur dengan biaya lebih rendah dari sebelumnya, maka biaya dalam antrian akan disesuaikan. Pendekatan ini memiliki kemiripan dengan algoritma Dijkstra, meskipun dalam UCS node baru dimasukkan secara dinamis saat eksplorasi berlangsung, bukan sekaligus di awal.

UCS mengevaluasi simpul menggunakan fungsi  $f(n) = g(n)$ , di mana  $g(n)$  merepresentasikan total biaya dari awal hingga simpul  $n$ . Selama bobot lintasan bersifat non-negatif, UCS akan selalu menghasilkan solusi yang optimal. Namun, dari segi performa, algoritma ini dapat menjadi lambat pada graf yang kompleks atau memiliki banyak jalur dengan biaya hampir serupa. Kompleksitas waktunya berada pada orde  $O(b^{(1 + C/\epsilon)})$ , di mana:

- $b$  adalah rata-rata jumlah cabang dari setiap node (branching factor),
- $C$  adalah total biaya solusi optimal,

- $\epsilon$  adalah bobot minimum dari setiap langkah.

Algoritma dalam notasi pseudocode:

```

function UNIFORM_COST_SEARCH(start, goal):
    // Inisialisasi struktur data
    openSet ← priority queue ordered by g(n)
    // g(n) = biaya kumulatif dari start ke n
    closedSet ← empty set

    gScore[start] ← 0
    openSet.enqueue(start, priority = gScore[start])

    while openSet is not empty:
        current ← openSet.dequeue()           // Ambil simpul dengan g terkecil

        if current == goal:
            return RECONSTRUCT_PATH(current)

        closedSet.add(current)

        for each move in current.getAvailableMoves():
            neighbor ← result of applying move to current
            tentative_g ← gScore[current] + cost(current, neighbor)

            if neighbor in closedSet:
                continue           // Lewati simpul yang sudah dieksplorasi

            if neighbor not in openSet or tentative_g < gScore[neighbor]:
                parent[neighbor] ← current
                gScore[neighbor] ← tentative_g

                if neighbor not in openSet:
                    openSet.enqueue(neighbor, priority = tentative_g)
                else:
                    openSet.updatePriority(neighbor, priority = tentative_g)

    // Jika openSet kosong tanpa menemukan goal
    return FAILURE

function RECONSTRUCT_PATH(node):
    path ← empty list
    while node has parent:
        path.prepend(node.moveFromParent)
        node ← parent[node]
    return path

```

## 2.2 Algoritma Greedy Best First Search (GBFS)

Greedy Best-First Search (GBFS) adalah metode pencarian yang memanfaatkan fungsi heuristik untuk memprioritaskan perluasan simpul yang terlihat paling dekat dengan tujuan. Alih-alih menghitung akumulasi biaya dari titik awal, GBFS hanya mempertimbangkan nilai  $h(n)$ , yaitu estimasi jarak atau biaya dari simpul  $n$  menuju sasaran. Dengan demikian, algoritma ini bergerak liar ke arah yang paling menjanjikan menurut prediksi heuristik, tanpa peduli seberapa mahal jalur yang sudah dilalui.

Proses GBFS dimulai dengan menempatkan status awal ke dalam struktur data open list (min-heap) yang diurutkan berdasarkan nilai heuristik terendah. Pada setiap iterasi, simpul dengan  $h(n)$  terkecil diambil dan diperiksa; apabila belum dikunjungi, maka ia ditandai sebagai telah dieksplorasi. Jika simpul tersebut memenuhi kondisi tujuan, pencarian selesai dan solusi dikembalikan. Jika tidak, semua tetangga yang valid akan dihitung nilai heuristik barunya dan selama biaya langkah ( $g(n)$ ) tidak melebihi batas maksimum, dimasukkan ke dalam open list untuk dievaluasi selanjutnya. Langkah ini berulang hingga ditemukan solusi atau tidak ada lagi simpul yang tersisa untuk dieksplorasi.

Secara umum, GBFS memiliki keunggulan dalam kecepatan dan penggunaan memori yang relatif hemat, karena hanya frontier (simpul yang sedang dipertimbangkan) dan himpunan visited yang perlu disimpan. Namun, kekuatan heuristik ini juga menjadi kelemahan. Jika fungsi heuristik kurang akurat, algoritma mudah terjebak di lembah lokal (local minima) atau dataran datar (plateau), dan tidak menjamin rute yang ditemukan adalah optimal. Selain itu, keputusan yang telah diambil bersifat final, algoritma tidak kembali meninjau ulang jalur yang pernah dilewati, sehingga potensi tersesat di jalur suboptimal cukup besar.

Algoritma dalam notasi pseudocode:

```
function GREEDY_BEST_FIRST_SEARCH(start, goal, HEURISTIC):
    // Inisialisasi
    openSet ← priority queue ordered by h(n)
    closedSet ← empty set

    hScore[start] ← HEURISTIC(start)
    openSet.enqueue(start, priority = hScore[start])
    parent[start] ← null

    while openSet is not empty:
        current ← openSet.dequeue()    // Ambil simpul dengan h(n) terkecil

        if current == goal:
            return RECONSTRUCT_PATH(current)

    closedSet.add(current)

    for each neighbor of current:
        if neighbor is not in closedSet:
            gScore[neighbor] ← gScore[current] + distance(current, neighbor)
            fScore[neighbor] ← gScore[neighbor] + hScore[neighbor]
            if neighbor is not in openSet:
                openSet.enqueue(neighbor, priority = fScore[neighbor])
                parent[neighbor] ← current
            else:
                if fScore[neighbor] > fScore[openSet.get(neighbor)]:
                    openSet.update(neighbor, priority = fScore[neighbor])
                    parent[neighbor] ← current
```

```

closedSet.add(current)

for each neighbor in current.getAvailableMoves():
    if neighbor in closedSet:
        continue           // Lewati simpul yang sudah dikunjungi

    hScore[neighbor] ← HEURISTIC(neighbor)

    if neighbor not in openSet:
        parent[neighbor] ← current
        openSet.enqueue(neighbor, priority = hScore[neighbor])
    // Jika neighbor sudah ada di openSet, kita tidak memperbarui
    // karena GBFS tidak mempertimbangkan g(n), hanya memilih
    // berdasarkan h(n)

return FAILURE

function RECONSTRUCT_PATH(node):
    path ← empty list
    while node has parent:
        path.prepend(node.moveFromParent)
        node ← parent[node]
    return path

```

### 2.3 Algoritma A\*

A\* adalah algoritma pencarian jalur terpendek yang memadukan dua pendekatan, yaitu mengakumulasi biaya nyata dari titik awal (seperti pada UCS) dan memanfaatkan estimasi jarak ke tujuan (seperti pada GBFS). Setiap simpul dinilai dengan fungsi  $f(n) = g(n) + h(n)$ , di mana  $g(n)$  adalah biaya yang sudah dikeluarkan dari awal sampai simpul  $n$ , dan  $h(n)$  adalah perkiraan biaya dari  $n$  menuju tujuan. Dengan cara ini, A\* tidak hanya berhitung secara teliti terhadap jalur yang sudah dilalui, tetapi juga memandang jauh ke depan untuk memilih jalur yang paling menjanjikan.

Kekuatan A\* terletak pada syarat heuristiknya. Jika fungsi heuristik  $h$  tidak pernah melebih-lebihkan biaya sebenarnya (admissible), maka A\* akan selalu menghasilkan solusi optimal. Apabila  $h$  juga memenuhi sifat konsistensi (monotonic), yaitu  $h(n) \leq c(n, m) + h(m)$  untuk setiap tetangga  $m$  dari  $n$ , maka A\* menjamin efisiensi lebih tinggi karena tidak perlu memproses ulang simpul-simpul yang sama. Dengan demikian, algoritma ini selalu menemukan solusi jika ada dan optimal dalam hal biaya rute.

Di balik keunggulannya, A\* memiliki kelemahan utama pada penggunaan memori dan waktu. Dalam kasus graf besar atau heuristik yang kurang informatif, jumlah simpul yang disimpan dan diperiksa bisa tumbuh secara eksponensial. Semua simpul yang sudah diekstrak maupun yang masih menunggu di open list disimpan dalam memori untuk mencegah pengulangan jalur. Oleh karena itu, meski A\* sangat akurat dan fleksibel, untuk ruang pencarian yang sangat luas atau terbatasnya memori, kinerjanya dapat menjadi kendala serius.

Algoritma dalam notasi pseudocode:

```
function A_STAR(start, goal, HEURISTIC):
    // Inisialisasi struktur data
    openSet ← priority queue ordered by f(n) = g(n) + h(n)
    closedSet ← empty set

    gScore[start] ← 0
    hScore[start] ← HEURISTIC(start, goal)
    fScore[start] ← gScore[start] + hScore[start]

    openSet.enqueue(start, fScore[start])

    while openSet is not empty:
        current ← openSet.dequeue() // Ambil simpul dengan f terkecil

        if current == goal:
            return RECONSTRUCT_PATH(current)

        closedSet.add(current)

        for each neighbor in current.getNeighbors():
            tentative_g ← gScore[current] + cost(current, neighbor)

            if neighbor in closedSet and tentative_g ≥ gScore[neighbor]:
                continue // Lewati jika sudah dievaluasi lebih baik

            if neighbor not in openSet or tentative_g < gScore[neighbor]:
                parent[neighbor] ← current
                gScore[neighbor] ← tentative_g
                hScore[neighbor] ← HEURISTIC(neighbor, goal)
                fScore[neighbor] ← gScore[neighbor] + hScore[neighbor]

                if neighbor not in openSet:
                    openSet.enqueue(neighbor, fScore[neighbor])
                else:
                    openSet.updatePriority(neighbor, fScore[neighbor])

    // Jika openSet habis tanpa menemukan goal
    return FAILURE
```

```

function RECONSTRUCT_PATH(node):
    path ← empty list
    while node has parent:
        path.prepend(node)
        node ← parent[node]
    path.prepend(start)
    return path

```

## 2.4 Algoritma Fringe Search

Fringe Search memadukan keunggulan memori yang ditawarkan IDA\* dengan keefektifan heuristik A\* melalui pendekatan iterative deepening pada nilai evaluasi total. Daripada mengelola satu frontier besar yang terus tumbuh seperti pada A\*, algoritma ini melakukan serangkaian gelombang pencarian di mana setiap gelombang hanya berkembang untuk simpul-simpul dengan  $f(n) = g(n) + h(n) \leq f_{\text{limit}}$ , di  $g(n)$  adalah biaya nyata dari titik awal hingga simpul  $n$ ,  $h(n)$  adalah estimasi admissible dari sisa biaya menuju tujuan.

Pada setiap iterasi,

1. Current Fringe berisi simpul-simpul yang  $f(n)$  tidak melebihi ambang  $f_{\text{limit}}$ .
2. Next Fringe menampung simpul-simpul dengan nilai  $f(n)$  lebih besar, sekaligus mencatat nilai  $f$  terkecil di antara mereka sebagai ambang berikutnya.

Simpul-simpul di Current Fringe kemudian dikembangkan satu per satu, mengikuti pola DFS, dengan setiap tetangga yang masih memenuhi batas  $f$  dimasukkan kembali ke Current Fringe, sedangkan tetangga lain dipindahkan ke Next Fringe. Setelah Current Fringe kosong, algoritma menaikkan batas  $f$  menjadi nilai minimum yang tercatat di Next Fringe, lalu memulai gelombang baru.

Dengan cara ini, penggunaan memori dibatasi pada dua daftar berukuran relatif kecil, bukan satu frontier yang besar. Fringe Search tetap optimal selama heuristik  $h$  bersifat admissible, dan mengekspansi lebih sedikit simpul ulang dibanding IDA\* karena tidak perlu mundur sepenuhnya setiap kali menaikkan ambang. Dari segi kompleksitas, waktu terburuknya bersifat eksponensial  $O(b^d)$ , namun ruang yang dibutuhkan hanya  $O(b \times m)$  dengan  $m$  kedalaman maksimum dalam setiap gelombang, jauh lebih rendah daripada  $O(b^d)$  frontier A\*.

Algoritma dalam notasi pseudocode:

```

function FRINGE_SEARCH(start, goal, HEURISTIC):
    // Inisialisasi state awal
    start.g ← 0
    start.h ← HEURISTIC(start)
    start.f ← start.g + start.h

    threshold ← start.f
    fringe ← [ start ]           // daftar state yang akan diperiksa
    cache ← { start: 0 }         // simpan g-score terendah untuk setiap state

    while fringe is not empty:
        nextThreshold ← ∞
        i ← 0

        // Iterasi melalui fringe sekaligus menambah successors
        while i < length(fringe):
            node ← fringe[i]

            if node.f > threshold:
                // Tunda ekspansi node ini; catat ambang minimum berikutnya
                nextThreshold ← min(nextThreshold, node.f)
                i ← i + 1
            else:
                // Ekspansi node yang memenuhi ambang f ≤ threshold
                remove fringe[i]           // keluarkan node dari fringe
                if node == goal:
                    return RECONSTRUCT_PATH(node)

                for each move in node.getAvailableMoves():
                    succ ← APPLY_MOVE(node, move)
                    succ.g ← node.g + cost(node, succ)
                    succ.h ← HEURISTIC(succ)
                    succ.f ← succ.g + succ.h

                    old_g ← cache.get(succ, ∞)
                    if succ.g < old_g:
                        cache[succ] ← succ.g
                        // Sisipkan succ ke fringe, urut berdasarkan f naik
                        INSERT_SORTED_BY_F(fringe, succ)
                    // (i tetap sama karena kita menggantikan posisi yang baru
                    // diisi)

                if nextThreshold == ∞:
                    break           // tidak ada state tersisa yang bisa diekspansi

            threshold ← nextThreshold

        return FAILURE // goal tidak ditemukan

function RECONSTRUCT_PATH(node):
    path ← empty list

```

```
while node has parent:  
    path.prepend(node.moveFromParent)  
    node ← node.parent  
return path
```

## **BAB III**

### **ANALISIS ALGORITMA**

#### **3.1 Fungsi Evaluasi $f(n)$ dan $g(n)$**

Semua algoritma jalur memerlukan cara untuk menentukan urutan pemrosesan simpul dengan mempertimbangkan nilai  $f(n)$ , yaitu kriteria keseluruhan yang dipakai frontier untuk memilih simpul berikutnya. Sementara itu, nilai  $g(n)$  menggambarkan biaya riil yang telah dikeluarkan dari titik awal hingga simpul  $n$ , misalnya, jumlah perpindahan mobil di Rush Hour sampai kondisi tertentu. Pada algoritma UCS cukup  $f(n)=g(n)$ , artinya hanya biaya riil yang diperhitungkan. Algoritma Greedy Best-First Search menetapkan  $f(n) = h(n)$ , hanya estimasi jarak ke tujuan sehingga jalur bisa sangat mahal karena memilih arah yang tampak paling dekat ke tujuan meskipun sebenarnya memutar atau tidak efisien. Algoritma A\* menggabungkan dua komponen tersebut, yaitu  $f(n) = g(n) + h(n)$  yang menyeimbangkan riil dan prediksi.

#### **3.2 Keadmissibilitasan Heuristik pada A\***

Dalam konteks A\*, sebuah fungsi heuristik dianggap admissible apabila estimasinya tidak pernah melebihi biaya sebenarnya yang diperlukan untuk mencapai tujuan, sehingga selalu “optimis” terhadap sisa jarak. setiap simpul  $n$  dinilai dengan fungsi  $f(n) = g(n) + h(n)$ , di mana  $g(n)$  adalah biaya kumulatif dari awal hingga  $n$  dan  $h(n)$  adalah estimasi biaya dari  $n$  ke tujuan. Heuristik dikatakan *admissible* jika untuk semua  $n$  berlaku  $h(n) \leq h^*(n)$ , di mana  $h^*(n)$  adalah biaya sebenarnya dari  $n$  ke tujuan.

Pada heuristik Manhattan distance, nilai yang dihasilkan hanya mengukur jumlah langkah minimal di sepanjang sumbu horizontal dan vertikal tanpa memperhitungkan rintangan apa pun, sehingga tidak mungkin lebih besar daripada jumlah gerakan riil yang diperlukan. Sementara itu, heuristik blocking cells menghitung berapa banyak kotak yang harus dikosongkan pada jalur menuju pintu keluar; karena setiap kotak penghalang setidaknya memerlukan satu perpindahan untuk dilewati, jumlah kotak tersebut menjadi batas bawah yang valid untuk langkah yang dibutuhkan. Terakhir, heuristik blocking vehicles mencerminkan jumlah kendaraan yang menghalangi lintasan, karena setiap kendaraan harus digeser setidaknya sekali agar jalan terbuka, nilai heuristik ini juga tidak akan melebih-lebihkan jumlah perpindahan aktual. Dengan demikian Manhattan distance, blocking cells, dan blocking vehicles, memenuhi syarat admissibility dalam algoritma A\*.

### **3.3 Perbandingan UCS dan BFS pada Penyelesaian Rush Hour**

Pada penyelesaian Rush Hour, karena setiap pergerakan kendaraan dihargai satu langkah, UCS yang menyortir frontier berdasarkan  $g(n)$  = kedalaman simpul, akan memperluas konfigurasi level-per-level persis seperti BFS. Artinya, baik UCS maupun BFS menjamin menemukan solusi dengan langkah minimal. Namun, secara implementasi UCS dan BFS berbeda. BFS menggunakan antrian FIFO, sehingga jika posisi A didorong sebelum B, BFS memproses A lalu B. UCS, di sisi lain, menggunakan priority queue (min-heap). Sebagai contoh, ketika dua konfigurasi memiliki  $g = 3$ , simpul mana yang diambil lebih dulu bergantung pada urutan internal heap dan aturan pemecahan seri (tie-breaking), misalnya UCS mungkin memproses konfigurasi C sebelum D, meski dalam BFS urutannya terbalik.

### **3.4 Efisiensi Algoritma A\* dibandingkan UCS pada Penyelesaian Rush Hour**

A\* lebih efisien dibandingkan Uniform Cost Search (UCS) dalam menyelesaikan permainan Rush Hour karena A\* menggunakan fungsi heuristik untuk memandu pencarian menuju keadaan tujuan. UCS hanya mengembangkan simpul berdasarkan nilai biaya aktual  $g(n)$  tanpa mempertimbangkan seberapa dekat simpul tersebut ke solusi. Sebaliknya, A\* memilih simpul berdasarkan nilai  $f(n) = g(n) + h(n)$ , di mana  $h(n)$  adalah estimasi biaya tersisa menuju tujuan yang bersifat admissible dan konsisten.

Dengan heuristik admissible, seperti jumlah kendaraan penghalang atau jarak Manhattan kunci, A\* dapat mengeliminasi cabang-cabang pencarian yang kurang menjanjikan sehingga jumlah simpul yang dihasilkan jauh lebih sedikit dibandingkan UCS. Walaupun keduanya tetap menjamin solusi optimal ketika  $h(n)$  admissible, A\* secara signifikan lebih cepat menemukan solusi karena setiap perluasan simpul mempertimbangkan estimasi total biaya  $f(n)$ , bukan sekadar biaya terakumulasi  $g(n)$ . Akibatnya, ukuran ruang pencarian yang dieksplorasi A\* pada puzzle berukuran besar seperti Rush Hour menjadi jauh lebih kecil, menjadikannya secara teoritis lebih efisien baik dari segi jumlah simpul yang diperiksa maupun waktu eksekusi.

### **3.5 Optimalitas Greedy Best First Search dalam Rush Hour**

Greedy Best-First Search seringkali sangat cepat menemukan solusi karena menelusuri simpul dengan nilai heuristik  $h(n)$  terendah terlebih dahulu tanpa pernah memperhitungkan biaya yang sudah terakumulasi, yaitu  $g(n)$ . Akibatnya, meski suatu simpul tampak dekat dengan tujuan menurut heuristik, biaya riil  $g(n)$  untuk mencapainya bisa saja sangat tinggi. Sebagai contoh, mobil merah yang tinggal satu

kotak lagi menuju pintu keluar ( $h(n) = 1$ ), tetapi untuk mencapai kotak tersebut, perlu menggeser dua kendaraan lain dengan total  $g(n) = 3$  langkah. GBFS akan terus mengejar simpul dengan  $h(n) = 1$  tersebut, kemudian baru menyadari dead-end dan berbalik yang akan mengakumulasi lebih banyak langkah karena  $g(n)$  tidak pernah menjadi pertimbangan. Dengan demikian, meski responsif, GBFS tidak menjamin minimasi  $g(n)$  secara keseluruhan dan seringkali jauh dari solusi optimal.

# BAB IV

## IMPLEMENTASI

### 4.1 Algoritma

Algoritma	Tampilan Kode
UCS	<pre>● ● ●  1 import { Board } from '../models/Board'; 2 import { Move } from '../models/Move'; 3 import { State } from '../models/State'; 4 import PriorityQueue from 'ts-priority-queue'; 5 import type { HeuristicFunction } from './heuristics/index'; 6 7 // Greedy Best First Search algorithm 8 export function GBFS( 9   initialBoard: Board, 10   heuristic: HeuristicFunction 11 ): { 12   solution: Move[], 13   nodesVisited: number, 14   executionTime: number 15 } { 16   const startTime = performance.now(); 17   let nodesVisited = 0; 18 19   const openSet = new PriorityQueue&lt;State&gt;({ 20     comparator: (a, b) =&gt; a.heuristicValue - b.heuristicValue 21   }); 22 23   const initialState = new State(initialBoard); 24   initialState.heuristicValue = heuristic(initialState.board); 25   openSet.queue(initialState); 26 27   const visitedStates = new Set&lt;string&gt;(); 28   visitedStates.add(initialState.hash()); 29 30   while (openSet.length &gt; 0) { 31     const current = openSet.dequeue(); 32     nodesVisited++; 33 34     if (current.board.isPuzzleSolved()) { 35       const endTime = performance.now(); 36       return { 37         solution: current.getPath(), 38         nodesVisited, 39         executionTime: endTime - startTime 40       }; 41     } 42 43     const availableMoves = current.board.getAvailableMoves(); 44 45     for (const move of availableMoves) { 46       const newBoard = current.board.applyMove(move); 47       const newState = new State(newBoard, current, move); 48 49       newState.heuristicValue = heuristic(newState); 50 51       const hash = newState.hash(); 52       if (!visitedStates.has(hash)) { 53         visitedStates.add(hash); 54         openSet.queue(newState); 55       } 56     } 57   } 58 59   const endTime = performance.now(); 60   return { 61     solution: [], 62     nodesVisited, 63     executionTime: endTime - startTime 64   }; 65 }</pre>

## GBFS

```
● ● ●
1 import { Board } from '../models/Board';
2 import { Move } from '../models/Move';
3 import { State } from '../models/State';
4 import PriorityQueue from 'ts-priority-queue';
5 import type { HeuristicFunction } from './heuristics/index';
6
7 // Greedy Best First Search algorithm
8 export function GBFS(
9   initialBoard: Board,
10   heuristic: HeuristicFunction
11 ): {
12   solution: Move[],
13   nodesVisited: number,
14   executionTime: number
15 } {
16   const startTime = performance.now();
17   let nodesVisited = 0;
18
19   const openSet = new PriorityQueue<State>({
20     comparator: (a, b) => a.heuristicValue - b.heuristicValue
21   });
22
23   const initialState = new State(initialBoard);
24   initialState.heuristicValue = heuristic(initialState.board);
25   openSet.queue(initialState);
26
27   const visitedStates = new Set<string>();
28   visitedStates.add(initialState.hash());
29
30   while (openSet.length > 0) {
31     const current = openSet.dequeue();
32     nodesVisited++;
33
34     if (current.board.isPuzzleSolved()) {
35       const endTime = performance.now();
36       return {
37         solution: current.getPath(),
38         nodesVisited,
39         executionTime: endTime - startTime
40       };
41     }
42
43     const availableMoves = current.board.getAvailableMoves();
44
45     for (const move of availableMoves) {
46       const newBoard = current.board.applyMove(move);
47       const newState = new State(newBoard, current, move);
48
49       newState.heuristicValue = heuristic(newState);
50
51       const hash = newState.hash();
52       if (!visitedStates.has(hash)) {
53         visitedStates.add(hash);
54         openSet.queue(newState);
55       }
56     }
57   }
58
59   const endTime = performance.now();
60   return {
61     solution: [],
62     nodesVisited,
63     executionTime: endTime - startTime
64   };
65 }
```

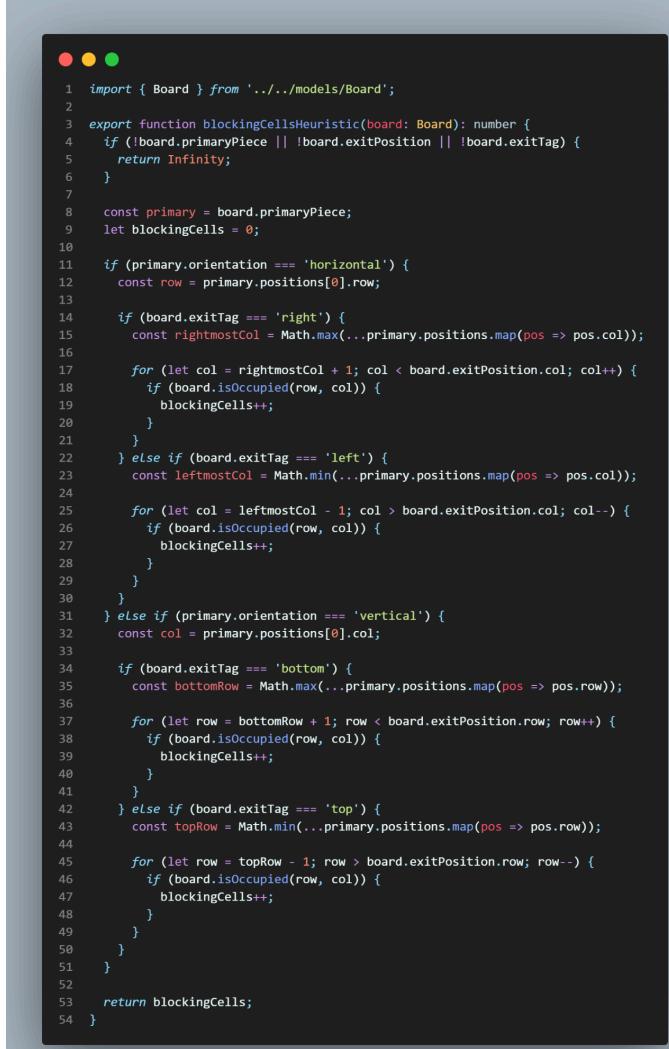
## A\*

```
1 import { Board } from '../models/Board';
2 import { Move } from '../models/Move';
3 import { State } from '../models/State';
4 import PriorityQueue from 'ts-priority-queue';
5 import type { HeuristicFunction } from '../algorithms/heuristics';
6
7 // A* Search algorithm
8 export function Astar(
9   initialBoard: Board,
10   heuristic: HeuristicFunction
11 ): {
12   solution: Move[],
13   nodesVisited: number,
14   executionTime: number
15 } {
16   const startTime = performance.now();
17   let nodesVisited = 0;
18
19   const openSet = new PriorityQueue<State>({
20     comparator: (a, b) => a.f - b.f
21   });
22
23   const initialState = new State(initialBoard);
24   initialState.heuristicValue = heuristic(initialState.board);
25   openSet.queue(initialState);
26
27   const visitedStates = new Map<string, number>();
28   visitedStates.set(initialState.hash(), initialState.cost);
29
30   while (openSet.length > 0) {
31     const current = openSet.dequeue();
32     nodesVisited++;
33
34     if (current.board.isPuzzleSolved()) {
35       const endTime = performance.now();
36       return {
37         solution: current.getPath(),
38         nodesVisited,
39         executionTime: endTime - startTime
40       };
41     }
42
43     const availableMoves = current.board.getAvailableMoves();
44
45     for (const move of availableMoves) {
46       const newBoard = current.board.applyMove(move);
47       const newState = new State(newBoard, current, move);
48
49       newState.heuristicValue = heuristic(newState);
50
51       const hash = newState.hash();
52       const existingCost = visitedStates.get(hash);
53
54       if (existingCost === undefined || newState.cost < existingCost) {
55         visitedStates.set(hash, newState.cost);
56         openSet.queue(newState);
57       }
58     }
59   }
60
61   const endTime = performance.now();
62   return {
63     solution: [],
64     nodesVisited,
65     executionTime: endTime - startTime
66   };
67 }
```

## Fringe Search

```
1 import { Board } from '../models/Board';
2 import { Move } from '../models/Move';
3 import { State } from '../models/State';
4 import type { HeuristicFunction } from './heuristics';
5
6 export function FringeSearch(
7   initialBoard: Board,
8   heuristic: HeuristicFunction
9 ): {
10   solution: Move[],
11   nodesVisited: number,
12   executionTime: number
13 } {
14   const startTime = performance.now();
15   let nodesVisited = 0;
16
17   const initialState = new State(initialBoard);
18   initialState.heuristicValue = heuristic(initialState.board);
19
20   let fringe: State[] = [initialState];
21
22   const cache = new Map<string, number>();
23   cache.set(initialState.hash(), initialState.cost);
24
25   let threshold = initialState.f;
26
27   let nextThreshold = Infinity;
28
29   while (fringe.length > 0) {
30     nextThreshold = Infinity;
31
32     let i = 0;
33
34     while (i < fringe.length) {
35       const current = fringe[i];
36       nodesVisited++;
37
38       if (current.board.isPuzzleSolved()) {
39         const endTime = performance.now();
40         return {
41           solution: current.getPath(),
42           nodesVisited,
43           executionTime: endTime - startTime
44         };
45       }
46
47       if (current.f > threshold) {
48         nextThreshold = Math.min(nextThreshold, current.f);
49         i++;
50         continue;
51       }
52
53       fringe.splice(i, 1);
54
55       const availableMoves = current.board.getAvailableMoves();
56       const successors: State[] = [];
57
58       for (const move of availableMoves) {
59         const newBoard = current.board.applyMove(move);
60         const newState = new State(newBoard, current, move);
61         newState.heuristicValue = heuristic(newState);
62
63         const hash = newState.hash();
64         const cachedCost = cache.get(hash);
65
66         if (cachedCost === undefined || newState.cost < cachedCost) {
67           cache.set(hash, newState.cost);
68           successors.push(newState);
69         }
70       }
71
72       successors.sort((a, b) => a.f - b.f);
73
74       fringe.splice(i, 0, ...successors);
75     }
76
77     if (nextThreshold === Infinity) {
78       break;
79     }
80
81     threshold = nextThreshold;
82   }
83
84   const endTime = performance.now();
85   return {
86     solution: [],
87     nodesVisited,
88     executionTime: endTime - startTime
89   };
90 }
```

## 4.2 Heuristik

Heuristik	Tampilan Kode
Blocking Cell	 <pre>import { Board } from '../../models/Board';  export function blockingCellsHeuristic(board: Board): number {     if (!board.primaryPiece    !board.exitPosition    !board.exitTag) {         return Infinity;     }      const primary = board.primaryPiece;     let blockingCells = 0;      if (primary.orientation === 'horizontal') {         const row = primary.positions[0].row;          if (board.exitTag === 'right') {             const rightmostCol = Math.max(...primary.positions.map(pos =&gt; pos.col));              for (let col = rightmostCol + 1; col &lt; board.exitPosition.col; col++) {                 if (board.isOccupied(row, col)) {                     blockingCells++;                 }             }         } else if (board.exitTag === 'left') {             const leftmostCol = Math.min(...primary.positions.map(pos =&gt; pos.col));              for (let col = leftmostCol - 1; col &gt; board.exitPosition.col; col--) {                 if (board.isOccupied(row, col)) {                     blockingCells++;                 }             }         } else if (primary.orientation === 'vertical') {             const col = primary.positions[0].col;              if (board.exitTag === 'bottom') {                 const bottomRow = Math.max(...primary.positions.map(pos =&gt; pos.row));                  for (let row = bottomRow + 1; row &lt; board.exitPosition.row; row++) {                     if (board.isOccupied(row, col)) {                         blockingCells++;                     }                 }             } else if (board.exitTag === 'top') {                 const topRow = Math.min(...primary.positions.map(pos =&gt; pos.row));                  for (let row = topRow - 1; row &gt; board.exitPosition.row; row--) {                     if (board.isOccupied(row, col)) {                         blockingCells++;                     }                 }             }         }     }      return blockingCells; }</pre>

## Blocking Vehicle

```
1 import { Board } from '../../../../../models/Board';
2
3 export function blockingVehiclesHeuristic(board: Board): number {
4   if (!board.primaryPiece || !board.exitPosition || !board.exitTag) {
5     return Infinity;
6   }
7
8   const primary = board.primaryPiece;
9   let blockingVehicles = 0;
10
11  if (primary.orientation === 'horizontal') {
12    const row = primary.positions[0].row;
13
14    if (board.exitTag === 'right') {
15      const rightmostCol = Math.max(...primary.positions.map(pos => pos.col));
16
17      for (let col = rightmostCol + 1; col < board.exitPosition.col; col++) {
18        const pieceAtPosition = board.getPieceAt(row, col);
19        if (pieceAtPosition && !pieceAtPosition.isPrimary) {
20          blockingVehicles++;
21          col += pieceAtPosition.size - 1;
22        }
23      }
24    } else if (board.exitTag === 'left') {
25      const leftmostCol = Math.min(...primary.positions.map(pos => pos.col));
26
27      for (let col = leftmostCol - 1; col > board.exitPosition.col; col--) {
28        const pieceAtPosition = board.getPieceAt(row, col);
29        if (pieceAtPosition && !pieceAtPosition.isPrimary) {
30          blockingVehicles++;
31          col -= pieceAtPosition.size - 1;
32        }
33      }
34    }
35  } else if (primary.orientation === 'vertical') {
36    const col = primary.positions[0].col;
37
38    if (board.exitTag === 'bottom') {
39      const bottomRow = Math.max(...primary.positions.map(pos => pos.row));
40
41      for (let row = bottomRow + 1; row < board.exitPosition.row; row++) {
42        const pieceAtPosition = board.getPieceAt(row, col);
43        if (pieceAtPosition && !pieceAtPosition.isPrimary) {
44          blockingVehicles++;
45          row += pieceAtPosition.size - 1;
46        }
47      }
48    } else if (board.exitTag === 'top') {
49      const topRow = Math.min(...primary.positions.map(pos => pos.row));
50
51      for (let row = topRow - 1; row > board.exitPosition.row; row--) {
52        const pieceAtPosition = board.getPieceAt(row, col);
53        if (pieceAtPosition && !pieceAtPosition.isPrimary) {
54          blockingVehicles++;
55          row -= pieceAtPosition.size - 1;
56        }
57      }
58    }
59  }
60
61  return blockingVehicles;
62 }
```

## Manhattan Distance

```
1 import { Board } from "../../models/Board";
2
3 export function manhattanDistanceHeuristic(board: Board): number {
4   if (!board.primaryPiece || !board.exitPosition || !board.exitTag) {
5     return Infinity;
6   }
7
8   const primary = board.primaryPiece;
9
10  if (primary.orientation === "horizontal") {
11    if (board.exitTag === "right") {
12      const rightmost = primary.getTail();
13      return Math.abs(rightmost.col - board.exitPosition.col);
14    } else if (board.exitTag === "left") {
15      const leftmost = primary.getHead();
16      return Math.abs(leftmost.col - board.exitPosition.col);
17    }
18  } else if (primary.orientation === "vertical") {
19    if (board.exitTag === "bottom") {
20      const bottommost = primary.getTail();
21      return Math.abs(bottommost.row - board.exitPosition.row);
22    } else if (board.exitTag === "top") {
23      const topmost = primary.getHead();
24      return Math.abs(topmost.row - board.exitPosition.row);
25    }
26  }
27
28  return Infinity;
29}
30
```

### 4.3 Model

Model	Tampilan Kode
-------	---------------

Board



## Move

```
1 import { Piece } from './Piece';
2 import type { Direction } from './Piece';
3
4 export class Move {
5   piece: Piece;
6   direction: Direction;
7   steps: number;
8
9   constructor(piece: Piece, direction: Direction, steps: number = 1) {
10     this.piece = piece;
11     this.direction = direction;
12     this.steps = steps;
13   }
14
15   toString(): string {
16     return `${this.piece.id}-${this.direction}`;
17   }
18 }
```

## Piece

```
1 export interface Position {
2   row: number;
3   col: number;
4 }
5
6 export type Orientation = 'horizontal' | 'vertical';
7 export type Direction = 'up' | 'down' | 'left' | 'right';
8
9 export class Piece {
10   id: string;
11   positions: Position[];
12   orientation: Orientation;
13   isPrimary: boolean;
14   size: number;
15
16   constructor(
17     id: string,
18     positions: Position[],
19     orientation: Orientation,
20     isPrimary: boolean = false
21   ) {
22     this.id = id;
23     this.positions = positions;
24     this.orientation = orientation;
25     this.isPrimary = isPrimary;
26     this.size = positions.length;
27   }
28
29   occupiesPosition(row: number, col: number): boolean {
30     return this.positions.some(pos => pos.row === row && pos.col === col);
31   }
32
33   getHead(): Position {
34     if (this.orientation === 'horizontal') {
35       return this.positions.reduce((min, pos) => (pos.col < min.col ? pos : min), this.positions[0]);
36     } else {
37       return this.positions.reduce((min, pos) => (pos.row < min.row ? pos : min), this.positions[0]);
38     }
39   }
40
41   getTail(): Position {
42     if (this.orientation === 'horizontal') {
43       return this.positions.reduce((max, pos) => (pos.col > max.col ? pos : max), this.positions[0]);
44     } else {
45       return this.positions.reduce((max, pos) => (pos.row > max.row ? pos : max), this.positions[0]);
46     }
47   }
48
49   clone(): Piece {
50     return new Piece(
51       this.id,
52       this.positions.map(pos => ({ row: pos.row, col: pos.col })),
53       this.orientation,
54       this.isPrimary
55     );
56   }
57 }
```

State	
	<pre> ● ● ●  1 import { Board } from './Board'; 2 import { Move } from './Move'; 3 4 export class State { 5   board: Board; 6   moves: Move[] = []; 7   parent: State   null = null; 8   cost: number = 0; 9   heuristicValue: number = 0; 10 11  constructor(board: Board, parent: State   null = null, lastMove: Move   null = null) { 12    this.board = board; 13    this.parent = parent; 14 15    if (parent &amp;&amp; lastMove) { 16      this.moves = [...parent.moves, lastMove]; 17      this.cost = parent.cost + 1; 18    } 19  } 20 21  get f(): number { 22    return this.cost + this.heuristicValue; 23  } 24 25  getPath(): Move[] { 26    return this.moves; 27  } 28 29  hash(): string { 30    return this.board.hash(); 31  } 32 }</pre>

#### 4.4 I/O

I/O	Tampilan Kode

fileParser

A screenshot of a terminal window with a black background and white text. The text is a long, continuous stream of characters, likely the output of a file parser. It appears to be a sequence of numbers, symbols, and possibly some structured data like JSON or XML. The terminal window has a dark border.

## saveSolution

```
● ● ●
1 import { Board } from '../models/Board';
2 import { Move } from '../models/Move';
3
4 export function generateSolutionText(boardStates: Board[], solution: Move[]): string {
5   let output = 'Papan Awal\n';
6   output += boardStates[0].toString();
7   output += '\n\n';
8
9   for (let i = 1; i < boardStates.length; i++) {
10     const board = boardStates[i];
11     const move = solution[i - 1];
12
13     let direction;
14     switch (move.direction) {
15       case 'up': direction = 'atas'; break;
16       case 'down': direction = 'bawah'; break;
17       case 'left': direction = 'kiri'; break;
18       case 'right': direction = 'kanan'; break;
19       default: direction = move.direction;
20     }
21
22     output += `Gerakan ${i}: ${move.piece.id}-${direction}\n`;
23     output += board.toString();
24     output += '\n\n';
25   }
26
27   return output;
28 }
29
30 export function downloadSolution(text: string, originalFilename: string = ''): void {
31   const blob = new Blob([text], { type: 'text/plain' });
32   const url = URL.createObjectURL(blob);
33
34   let filename;
35   if (originalFilename) {
36     const baseFilename = originalFilename.split('/').pop()?.split('\\').pop()?.split('.')[0] || 'puzzle';
37     filename = `${baseFilename}_solution.txt`;
38   } else {
39     filename = `rush-hour-solution-${Date.now()}.txt`;
40   }
41
42   const link = document.createElement('a');
43   link.href = url;
44   link.download = filename;
45
46   document.body.appendChild(link);
47   link.click();
48
49   document.body.removeChild(link);
50
51 }
```

## 4.5 Komponen

Komponen	Tampilan Kode
----------	---------------

## Board

```
● ● ●
1 import React from "react";
2 import { Board as BoardModel } from "../models/Board";
3 import Cell from "./Cell";
4
5 interface BoardProps {
6   board: BoardModel;
7   movedPieceId?: string;
8 }
9
10 const Board: React.FC<BoardProps> = ({ board, movedPieceId }) => {
11   let effectiveRows = board.rows;
12   let effectiveColumns = board.cols;
13
14   if (board.exitPosition) {
15     if (board.exitTag === "top" || board.exitTag === "bottom") {
16       effectiveRows++;
17     } else if (board.exitTag === "left" || board.exitTag === "right") {
18       effectiveColumns++;
19     }
20   }
21
22   const grid = Array(effectiveRows)
23     .fill(null)
24     .map(() => Array(effectiveColumns).fill(null));
25
26   let exitRow = -1;
27   let exitCol = -1;
28
29   if (board.exitPosition) {
30     exitRow = board.exitPosition.row;
31     exitCol = board.exitPosition.col;
32
33     if (
34       exitRow >= 0 &&
35       exitRow < effectiveRows &&
36       exitCol >= 0 &&
37       exitCol < effectiveColumns
38     ) {
39       grid[exitRow][exitCol] = [
40         id: "K",
41         isExit: true,
42         isPrimary: false,
43         isMoved: false,
44       ];
45     }
46   }
47
48   board.pieces.forEach((piece) => {
49     piece.positions.forEach((pos) => {
50       const row = pos.row;
51       const col = pos.col;
52
53       if (
54         row >= 0 &&
55         row < effectiveRows &&
56         col >= 0 &&
57         col < effectiveColumns
58       ) {
59         grid[row][col] = {
60           id: piece.id,
61           isPrimary: piece.isPrimary,
62           isMoved: piece.id === movedPieceId,
63           isExit: false,
64         };
65       }
66     });
67   });
68
69   const isExitRow = (rowIndex: number): boolean =>
70     (board.exitTag === "top" && rowIndex === 0) ||
71     (board.exitTag === "bottom" && rowIndex === effectiveRows - 1);
72
73   const isExitCol = (colIndex: number): boolean =>
74     (board.exitTag === "left" && colIndex === 0) ||
75     (board.exitTag === "right" && colIndex === effectiveColumns - 1);
76
77   const isExitPathCell = (rowIndex: number, colIndex: number): boolean => {
78     if (!board.exitPosition) return false;
79
80     if (board.exitTag === "top" && colIndex === exitCol && rowIndex === 0) return true;
81     if (board.exitTag === "bottom" && colIndex === exitCol && rowIndex === effectiveRows - 1) return true;
82     if (board.exitTag === "left" && rowIndex === exitRow && colIndex === 0) return true;
83     if (board.exitTag === "right" && rowIndex === exitRow && colIndex === effectiveColumns - 1) return true;
84
85     return false;
86   };
87
88   return (
89     <div className="inline-block border border-gray-700 bg-gray-100 my-5 p-4 rounded">
90       {grid.map((row, rowIndex) => (
91         <div key={rowIndex} className="flex">
92           {row.map((cell, colIndex) => (
93             <Cell
94               key={`${rowIndex}-${colIndex}`}
95               content={(cell ? cell.id : "")}
96               isPrimary={(cell?.isPrimary || false)}
97               isExit={(cell.isExit || false)}
98               isMoved={(cell.isMoved || false)}
99               isExitPath={isExitPathCell(rowIndex, colIndex)}
100               isExitRow={isExitRow(rowIndex)} && !isExitPathCell(rowIndex, colIndex)
101               isExitCol={isExitCol(colIndex)} && !isExitPathCell(rowIndex, colIndex)
102             />
103           )));
104         </div>
105       )));
106     </div>
107   );
108 };
109
110 export default Board;
```

## Cell

```
 1 import React from "react";
 2
 3 interface CellProps {
 4   content: string;
 5   isPrimary: boolean;
 6   isExit: boolean;
 7   isMoved: boolean;
 8   isExitPath: boolean;
 9   isExitRow?: boolean;
10   isExitColumn?: boolean;
11 }
12
13 const Cell: React.FC<CellProps> = ({(
14   content,
15   isPrimary,
16   isExit,
17   isMoved,
18   isExitPath,
19   isExitRow,
20   isExitColumn,
21 )} => {
22   let cellClasses =
23     "w-10 h-10 flex items-center justify-center font-bold text-lg border border-gray-300 box-border";
24
25   if (isExit) {
26     cellClasses += " bg-emerald-600 text-white";
27   } else if (isPrimary) {
28     cellClasses += " bg-red-600 text-white";
29   } else if (content === "" && content !== ".") {
30     const pieceColorMap: Record<string, string> = {
31       A: "bg-blue-500",
32       B: "bg-purple-500",
33       C: "bg-orange-500",
34       D: "bg-teal-500",
35       E: "bg-pink-500",
36       F: "bg-indigo-500",
37       G: "bg-amber-500",
38       H: "bg-cyan-500",
39       I: "bg-rose-500",
40       J: "bg-lime-500",
41       K: "bg-emerald-500",
42       L: "bg-sky-500",
43       M: "bg-fuchsia-500",
44       N: "bg-violet-500",
45       O: "bg-yellow-500",
46       P: "bg-blue-600",
47       Q: "bg-purple-600",
48       R: "bg-orange-600",
49       S: "bg-teal-600",
50       T: "bg-pink-600",
51       U: "bg-indigo-600",
52       V: "bg-amber-600",
53       W: "bg-cyan-600",
54       X: "bg-rose-600",
55       Y: "bg-lime-600",
56       Z: "bg-emerald-600",
57     };
58
59     cellClasses += ` ${pieceColorMap[content]} || "bg-gray-400" text-white`;
60   }
61
62   if (isMoved) {
63     cellClasses += " ring-4 ring-yellow-300 ring-inset";
64   }
65
66   if (isExitPath || isExitRow || isExitColumn) {
67     if (content === "" && content !== ".") {
68       cellClasses += " ring-2 ring-black ring-inset brightness-100";
69     } else {
70       cellClasses += " bg-gray-600 text-white";
71     }
72   }
73
74   return <div className={cellClasses}>{content}</div>;
75 };
76
77 export default Cell;
78
```

## ControlPanel

## FileInput

Footer



## Navbar

```
1 const Navbar = () => {
2   return (
3     <nav>
4       <div className="flex items-center w-full justify-between p-4 bg-black text-white">
5         <div className="text-lg font-bold text-pink-500">Rush Hour Solver</div>
6         <ul className="flex space-x-4">
7           <li><a href="/" className="hover:text-pink-400">Home</a></li>
8           <li><a href="/about" className="hover:text-pink-400">About</a></li>
9         </ul>
10       </div>
11     </nav>
12   )
13 }
14
15 export default Navbar
```

## SolutionDisplay

4.6 Pages

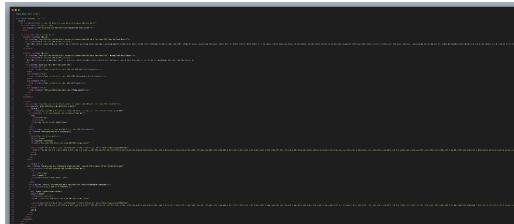
Pages	Tampilan Kode
-------	---------------

## Home



```
1 import React from "react";
2 import App from "../App";
3
4 const Home: React.FC = () => {
5   return <App />;
6 };
7
8 export default Home;
```

## About



## App

```
1 // Declarasi dan Inisialisasi
2 import {Board, update} from './components/tiling';
3 import {TilingRow} from './components/tilingRow';
4 import {GridCell} from './components/gridCell';
5 import {WordModel} from './models/wordModel';
6 import {SudokuModel} from './models/sudoku';
7 import {SubSudokuModel} from './models/subSudoku';
8 import {Algorithm} from './algorithms';
9 import {Huristic} from './heuristics';
10 import {Type} from './types';
11 import {AlgorithmType} from './algorithms/algorithm';
12 import {HuristicType} from './heuristics/heuristic';
13 import {TypeOf} from './algos/heuristic';
14
15 const [board, subboard] = update(createBoard([null, null]));
16 const [rows, columns] = estimateBoard([null, null]);
17 const [rowsX, columnsX] = estimateBoard([null, null]);
18 const [rowsY, columnsY] = estimateBoard([null, null]);
19 const [rowsZ, columnsZ] = estimateBoard([null, null]);
20
21 const orientationX = 'horizontal', orientationY = 'vertical';
22 const orientationZ = 'vertical', orientationW = 'horizontal';
23
24 const [rows, columns] = parseBoard([null, null], rows);
25
26 console.log('Rows: ', parsedBoard.rows);
27 console.log('Columns: ', parsedBoard.columns);
28
29 if (parsedBoard.numberOfTiles === parsedBoard.tiles.length - 1) {
30   setNumber(parsedBoard.numberOfTiles);
31 }
32
33 if (parsedBoard.orientationX) {
34   if (parsedBoard.orientationX === 'vertical') {
35     setOrientationX('vertical');
36   } else {
37     setOrientationX('horizontal');
38   }
39 }
40
41 if (parsedBoard.orientationY) {
42   if (parsedBoard.orientationY === 'horizontal') {
43     setOrientationY('horizontal');
44   } else {
45     setOrientationY('vertical');
46   }
47 }
48
49 if (parsedBoard.orientationZ) {
50   if (parsedBoard.orientationZ === 'vertical') {
51     setOrientationZ('vertical');
52   } else {
53     setOrientationZ('horizontal');
54   }
55 }
56
57 if (parsedBoard.orientationW) {
58   if (parsedBoard.orientationW === 'horizontal') {
59     setOrientationW('horizontal');
60   } else {
61     setOrientationW('vertical');
62   }
63 }
64
65 const boardSize = rows * columns;
66
67 if (rows < 3 || columns < 3) {
68   console.error(`Rows or columns must be at least 3!`);
69 }
70
71 if (rows * columns !== boardSize) {
72   console.error(`Rows and columns must multiply to ${boardSize}`);
73 }
74
75 if (rows % 3 === 0) {
76   if (rows / 3 % 3 === 0) {
77     setSubBoard();
78   } else {
79     setSubBoard();
80   }
81 }
82
83 if (rows % 3 === 1) {
84   if (rows / 3 % 3 === 0) {
85     setSubBoard();
86   } else {
87     setSubBoard();
88   }
89 }
90
91 if (rows % 3 === 2) {
92   if (rows / 3 % 3 === 0) {
93     setSubBoard();
94   } else {
95     setSubBoard();
96   }
97 }
98
99 if (rows % 3 === 0) {
100   if (rows / 3 % 3 === 0) {
101     setSubBoard();
102   } else {
103     setSubBoard();
104   }
105 }
106
107 if (rows % 3 === 1) {
108   if (rows / 3 % 3 === 0) {
109     setSubBoard();
110   } else {
111     setSubBoard();
112   }
113 }
114
115 if (rows % 3 === 2) {
116   if (rows / 3 % 3 === 0) {
117     setSubBoard();
118   } else {
119     setSubBoard();
120   }
121 }
122
123 if (rows % 3 === 0) {
124   if (rows / 3 % 3 === 0) {
125     setSubBoard();
126   } else {
127     setSubBoard();
128   }
129 }
130
131 if (rows % 3 === 1) {
132   if (rows / 3 % 3 === 0) {
133     setSubBoard();
134   } else {
135     setSubBoard();
136   }
137 }
138
139 if (rows % 3 === 2) {
140   if (rows / 3 % 3 === 0) {
141     setSubBoard();
142   } else {
143     setSubBoard();
144   }
145 }
146
147 if (rows % 3 === 0) {
148   if (rows / 3 % 3 === 0) {
149     setSubBoard();
150   } else {
151     setSubBoard();
152   }
153 }
154
155 if (rows % 3 === 1) {
156   if (rows / 3 % 3 === 0) {
157     setSubBoard();
158   } else {
159     setSubBoard();
160   }
161 }
162
163 if (rows % 3 === 2) {
164   if (rows / 3 % 3 === 0) {
165     setSubBoard();
166   } else {
167     setSubBoard();
168   }
169 }
170
171 if (rows % 3 === 0) {
172   if (rows / 3 % 3 === 0) {
173     setSubBoard();
174   } else {
175     setSubBoard();
176   }
177 }
178
179 if (rows % 3 === 1) {
180   if (rows / 3 % 3 === 0) {
181     setSubBoard();
182   } else {
183     setSubBoard();
184   }
185 }
186
187 if (rows % 3 === 2) {
188   if (rows / 3 % 3 === 0) {
189     setSubBoard();
190   } else {
191     setSubBoard();
192   }
193 }
194
195 if (rows % 3 === 0) {
196   if (rows / 3 % 3 === 0) {
197     setSubBoard();
198   } else {
199     setSubBoard();
200   }
201 }
202
203 if (rows % 3 === 1) {
204   if (rows / 3 % 3 === 0) {
205     setSubBoard();
206   } else {
207     setSubBoard();
208   }
209 }
210
211 if (rows % 3 === 2) {
212   if (rows / 3 % 3 === 0) {
213     setSubBoard();
214   } else {
215     setSubBoard();
216   }
217 }
218
219 if (rows % 3 === 0) {
220   if (rows / 3 % 3 === 0) {
221     setSubBoard();
222   } else {
223     setSubBoard();
224   }
225 }
226
227 if (rows % 3 === 1) {
228   if (rows / 3 % 3 === 0) {
229     setSubBoard();
230   } else {
231     setSubBoard();
232   }
233 }
234
235 if (rows % 3 === 2) {
236   if (rows / 3 % 3 === 0) {
237     setSubBoard();
238   } else {
239     setSubBoard();
240   }
241 }
242
243 if (rows % 3 === 0) {
244   if (rows / 3 % 3 === 0) {
245     setSubBoard();
246   } else {
247     setSubBoard();
248   }
249 }
250
251 if (rows % 3 === 1) {
252   if (rows / 3 % 3 === 0) {
253     setSubBoard();
254   } else {
255     setSubBoard();
256   }
257 }
258
259 if (rows % 3 === 2) {
260   if (rows / 3 % 3 === 0) {
261     setSubBoard();
262   } else {
263     setSubBoard();
264   }
265 }
266
267 if (rows % 3 === 0) {
268   if (rows / 3 % 3 === 0) {
269     setSubBoard();
270   } else {
271     setSubBoard();
272   }
273 }
274
275 if (rows % 3 === 1) {
276   if (rows / 3 % 3 === 0) {
277     setSubBoard();
278   } else {
279     setSubBoard();
280   }
281 }
282
283 if (rows % 3 === 2) {
284   if (rows / 3 % 3 === 0) {
285     setSubBoard();
286   } else {
287     setSubBoard();
288   }
289 }
290
291 if (rows % 3 === 0) {
292   if (rows / 3 % 3 === 0) {
293     setSubBoard();
294   } else {
295     setSubBoard();
296   }
297 }
298
299 if (rows % 3 === 1) {
300   if (rows / 3 % 3 === 0) {
301     setSubBoard();
302   } else {
303     setSubBoard();
304   }
305 }
306
307 if (rows % 3 === 2) {
308   if (rows / 3 % 3 === 0) {
309     setSubBoard();
310   } else {
311     setSubBoard();
312   }
313 }
314
315 if (rows % 3 === 0) {
316   if (rows / 3 % 3 === 0) {
317     setSubBoard();
318   } else {
319     setSubBoard();
320   }
321 }
322
323 if (rows % 3 === 1) {
324   if (rows / 3 % 3 === 0) {
325     setSubBoard();
326   } else {
327     setSubBoard();
328   }
329 }
330
331 if (rows % 3 === 2) {
332   if (rows / 3 % 3 === 0) {
333     setSubBoard();
334   } else {
335     setSubBoard();
336   }
337 }
338
339 if (rows % 3 === 0) {
340   if (rows / 3 % 3 === 0) {
341     setSubBoard();
342   } else {
343     setSubBoard();
344   }
345 }
346
347 if (rows % 3 === 1) {
348   if (rows / 3 % 3 === 0) {
349     setSubBoard();
350   } else {
351     setSubBoard();
352   }
353 }
354
355 if (rows % 3 === 2) {
356   if (rows / 3 % 3 === 0) {
357     setSubBoard();
358   } else {
359     setSubBoard();
360   }
361 }
362
363 if (rows % 3 === 0) {
364   if (rows / 3 % 3 === 0) {
365     setSubBoard();
366   } else {
367     setSubBoard();
368   }
369 }
370
371 if (rows % 3 === 1) {
372   if (rows / 3 % 3 === 0) {
373     setSubBoard();
374   } else {
375     setSubBoard();
376   }
377 }
378
379 if (rows % 3 === 2) {
380   if (rows / 3 % 3 === 0) {
381     setSubBoard();
382   } else {
383     setSubBoard();
384   }
385 }
386
387 if (rows % 3 === 0) {
388   if (rows / 3 % 3 === 0) {
389     setSubBoard();
390   } else {
391     setSubBoard();
392   }
393 }
394
395 if (rows % 3 === 1) {
396   if (rows / 3 % 3 === 0) {
397     setSubBoard();
398   } else {
399     setSubBoard();
400   }
401 }
402
403 if (rows % 3 === 2) {
404   if (rows / 3 % 3 === 0) {
405     setSubBoard();
406   } else {
407     setSubBoard();
408   }
409 }
410
411 if (rows % 3 === 0) {
412   if (rows / 3 % 3 === 0) {
413     setSubBoard();
414   } else {
415     setSubBoard();
416   }
417 }
418
419 if (rows % 3 === 1) {
420   if (rows / 3 % 3 === 0) {
421     setSubBoard();
422   } else {
423     setSubBoard();
424   }
425 }
426
427 if (rows % 3 === 2) {
428   if (rows / 3 % 3 === 0) {
429     setSubBoard();
430   } else {
431     setSubBoard();
432   }
433 }
434
435 if (rows % 3 === 0) {
436   if (rows / 3 % 3 === 0) {
437     setSubBoard();
438   } else {
439     setSubBoard();
440   }
441 }
442
443 if (rows % 3 === 1) {
444   if (rows / 3 % 3 === 0) {
445     setSubBoard();
446   } else {
447     setSubBoard();
448   }
449 }
450
451 if (rows % 3 === 2) {
452   if (rows / 3 % 3 === 0) {
453     setSubBoard();
454   } else {
455     setSubBoard();
456   }
457 }
458
459 if (rows % 3 === 0) {
460   if (rows / 3 % 3 === 0) {
461     setSubBoard();
462   } else {
463     setSubBoard();
464   }
465 }
466
467 if (rows % 3 === 1) {
468   if (rows / 3 % 3 === 0) {
469     setSubBoard();
470   } else {
471     setSubBoard();
472   }
473 }
474
475 if (rows % 3 === 2) {
476   if (rows / 3 % 3 === 0) {
477     setSubBoard();
478   } else {
479     setSubBoard();
480   }
481 }
482
483 if (rows % 3 === 0) {
484   if (rows / 3 % 3 === 0) {
485     setSubBoard();
486   } else {
487     setSubBoard();
488   }
489 }
490
491 if (rows % 3 === 1) {
492   if (rows / 3 % 3 === 0) {
493     setSubBoard();
494   } else {
495     setSubBoard();
496   }
497 }
498
499 if (rows % 3 === 2) {
500   if (rows / 3 % 3 === 0) {
501     setSubBoard();
502   } else {
503     setSubBoard();
504   }
505 }
506
507 if (rows % 3 === 0) {
508   if (rows / 3 % 3 === 0) {
509     setSubBoard();
510   } else {
511     setSubBoard();
512   }
513 }
514
515 if (rows % 3 === 1) {
516   if (rows / 3 % 3 === 0) {
517     setSubBoard();
518   } else {
519     setSubBoard();
520   }
521 }
522
523 if (rows % 3 === 2) {
524   if (rows / 3 % 3 === 0) {
525     setSubBoard();
526   } else {
527     setSubBoard();
528   }
529 }
530
531 if (rows % 3 === 0) {
532   if (rows / 3 % 3 === 0) {
533     setSubBoard();
534   } else {
535     setSubBoard();
536   }
537 }
538
539 if (rows % 3 === 1) {
540   if (rows / 3 % 3 === 0) {
541     setSubBoard();
542   } else {
543     setSubBoard();
544   }
545 }
546
547 if (rows % 3 === 2) {
548   if (rows / 3 % 3 === 0) {
549     setSubBoard();
550   } else {
551     setSubBoard();
552   }
553 }
554
555 if (rows % 3 === 0) {
556   if (rows / 3 % 3 === 0) {
557     setSubBoard();
558   } else {
559     setSubBoard();
560   }
561 }
562
563 if (rows % 3 === 1) {
564   if (rows / 3 % 3 === 0) {
565     setSubBoard();
566   } else {
567     setSubBoard();
568   }
569 }
570
571 if (rows % 3 === 2) {
572   if (rows / 3 % 3 === 0) {
573     setSubBoard();
574   } else {
575     setSubBoard();
576   }
577 }
578
579 if (rows % 3 === 0) {
580   if (rows / 3 % 3 === 0) {
581     setSubBoard();
582   } else {
583     setSubBoard();
584   }
585 }
586
587 if (rows % 3 === 1) {
588   if (rows / 3 % 3 === 0) {
589     setSubBoard();
590   } else {
591     setSubBoard();
592   }
593 }
594
595 if (rows % 3 === 2) {
596   if (rows / 3 % 3 === 0) {
597     setSubBoard();
598   } else {
599     setSubBoard();
600   }
601 }
602
603 if (rows % 3 === 0) {
604   if (rows / 3 % 3 === 0) {
605     setSubBoard();
606   } else {
607     setSubBoard();
608   }
609 }
610
611 if (rows % 3 === 1) {
612   if (rows / 3 % 3 === 0) {
613     setSubBoard();
614   } else {
615     setSubBoard();
616   }
617 }
618
619 if (rows % 3 === 2) {
620   if (rows / 3 % 3 === 0) {
621     setSubBoard();
622   } else {
623     setSubBoard();
624   }
625 }
626
627 if (rows % 3 === 0) {
628   if (rows / 3 % 3 === 0) {
629     setSubBoard();
630   } else {
631     setSubBoard();
632   }
633 }
634
635 if (rows % 3 === 1) {
636   if (rows / 3 % 3 === 0) {
637     setSubBoard();
638   } else {
639     setSubBoard();
640   }
641 }
642
643 if (rows % 3 === 2) {
644   if (rows / 3 % 3 === 0) {
645     setSubBoard();
646   } else {
647     setSubBoard();
648   }
649 }
650
651 if (rows % 3 === 0) {
652   if (rows / 3 % 3 === 0) {
653     setSubBoard();
654   } else {
655     setSubBoard();
656   }
657 }
658
659 if (rows % 3 === 1) {
660   if (rows / 3 % 3 === 0) {
661     setSubBoard();
662   } else {
663     setSubBoard();
664   }
665 }
666
667 if (rows % 3 === 2) {
668   if (rows / 3 % 3 === 0) {
669     setSubBoard();
670   } else {
671     setSubBoard();
672   }
673 }
674
675 if (rows % 3 === 0) {
676   if (rows / 3 % 3 === 0) {
677     setSubBoard();
678   } else {
679     setSubBoard();
680   }
681 }
682
683 if (rows % 3 === 1) {
684   if (rows / 3 % 3 === 0) {
685     setSubBoard();
686   } else {
687     setSubBoard();
688   }
689 }
690
691 if (rows % 3 === 2) {
692   if (rows / 3 % 3 === 0) {
693     setSubBoard();
694   } else {
695     setSubBoard();
696   }
697 }
698
699 if (rows % 3 === 0) {
700   if (rows / 3 % 3 === 0) {
701     setSubBoard();
702   } else {
703     setSubBoard();
704   }
705 }
706
707 if (rows % 3 === 1) {
708   if (rows / 3 % 3 === 0) {
709     setSubBoard();
710   } else {
711     setSubBoard();
712   }
713 }
714
715 if (rows % 3 === 2) {
716   if (rows / 3 % 3 === 0) {
717     setSubBoard();
718   } else {
719     setSubBoard();
720   }
721 }
722
723 if (rows % 3 === 0) {
724   if (rows / 3 % 3 === 0) {
725     setSubBoard();
726   } else {
727     setSubBoard();
728   }
729 }
730
731 if (rows % 3 === 1) {
732   if (rows / 3 % 3 === 0) {
733     setSubBoard();
734   } else {
735     setSubBoard();
736   }
737 }
738
739 if (rows % 3 === 2) {
740   if (rows / 3 % 3 === 0) {
741     setSubBoard();
742   } else {
743     setSubBoard();
744   }
745 }
746
747 if (rows % 3 === 0) {
748   if (rows / 3 % 3 === 0) {
749     setSubBoard();
750   } else {
751     setSubBoard();
752   }
753 }
754
755 if (rows % 3 === 1) {
756   if (rows / 3 % 3 === 0) {
757     setSubBoard();
758   } else {
759     setSubBoard();
760   }
761 }
762
763 if (rows % 3 === 2) {
764   if (rows / 3 % 3 === 0) {
765     setSubBoard();
766   } else {
767     setSubBoard();
768   }
769 }
770
771 if (rows % 3 === 0) {
772   if (rows / 3 % 3 === 0) {
773     setSubBoard();
774   } else {
775     setSubBoard();
776   }
777 }
778
779 if (rows % 3 === 1) {
780   if (rows / 3 % 3 === 0) {
781     setSubBoard();
782   } else {
783     setSubBoard();
784   }
785 }
786
787 if (rows % 3 === 2) {
788   if (rows / 3 % 3 === 0) {
789     setSubBoard();
790   } else {
791     setSubBoard();
792   }
793 }
794
795 if (rows % 3 === 0) {
796   if (rows / 3 % 3 === 0) {
797     setSubBoard();
798   } else {
799     setSubBoard();
800   }
801 }
802
803 if (rows % 3 === 1) {
804   if (rows / 3 % 3 === 0) {
805     setSubBoard();
806   } else {
807     setSubBoard();
808   }
809 }
810
811 if (rows % 3 === 2) {
812   if (rows / 3 % 3 === 0) {
813     setSubBoard();
814   } else {
815     setSubBoard();
816   }
817 }
818
819 if (rows % 3 === 0) {
820   if (rows / 3 % 3 === 0) {
821     setSubBoard();
822   } else {
823     setSubBoard();
824   }
825 }
826
827 if (rows % 3 === 1) {
828   if (rows / 3 % 3 === 0) {
829     setSubBoard();
830   } else {
831     setSubBoard();
832   }
833 }
834
835 if (rows % 3 === 2) {
836   if (rows / 3 % 3 === 0) {
837     setSubBoard();
838   } else {
839     setSubBoard();
840   }
841 }
842
843 if (rows % 3 === 0) {
844   if (rows / 3 % 3 === 0) {
845     setSubBoard();
846   } else {
847     setSubBoard();
848   }
849 }
850
851 if (rows % 3 === 1) {
852   if (rows / 3 % 3 === 0) {
853     setSubBoard();
854   } else {
855     setSubBoard();
856   }
857 }
858
859 if (rows % 3 === 2) {
860   if (rows / 3 % 3 === 0) {
861     setSubBoard();
862   } else {
863     setSubBoard();
864   }
865 }
866
867 if (rows % 3 === 0) {
868   if (rows / 3 % 3 === 0) {
869     setSubBoard();
870   } else {
871     setSubBoard();
872   }
873 }
874
875 if (rows % 3 === 1) {
876   if (rows / 3 % 3 === 0) {
877     setSubBoard();
878   } else {
879     setSubBoard();
880   }
881 }
882
883 if (rows % 3 === 2) {
884   if (rows / 3 % 3 === 0) {
885     setSubBoard();
886   } else {
887     setSubBoard();
888   }
889 }
890
891 if (rows % 3 === 0) {
892   if (rows / 3 % 3 === 0) {
893     setSubBoard();
894   } else {
895     setSubBoard();
896   }
897 }
898
899 if (rows % 3 === 1) {
900   if (rows / 3 % 3 === 0) {
901     setSubBoard();
902   } else {
903     setSubBoard();
904   }
905 }
906
907 if (rows % 3 === 2) {
908   if (rows / 3 % 3 === 0) {
909     setSubBoard();
910   } else {
911     setSubBoard();
912   }
913 }
914
915 if (rows % 3 === 0) {
916   if (rows / 3 % 3 === 0) {
917     setSubBoard();
918   } else {
919     setSubBoard();
920   }
921 }
922
923 if (rows % 3 === 1) {
924   if (rows / 3 % 3 === 0) {
925     setSubBoard();
926   } else {
927     setSubBoard();
928   }
929 }
930
931 if (rows % 3 === 2) {
932   if (rows / 3 % 3 === 0) {
933     setSubBoard();
934   } else {
935     setSubBoard();
936   }
937 }
938
939 if (rows % 3 === 0) {
940   if (rows / 3 % 3 === 0) {
941     setSubBoard();
942   } else {
943     setSubBoard();
944   }
945 }
946
947 if (rows % 3 === 1) {
948   if (rows / 3 % 3 === 0) {
949     setSubBoard();
950   } else {
951     setSubBoard();
952   }
953 }
954
955 if (rows % 3 === 2) {
956   if (rows / 3 % 3 === 0) {
957     setSubBoard();
958   } else {
959     setSubBoard();
960   }
961 }
962
963 if (rows % 3 === 0) {
964   if (rows / 3 % 3 === 0) {
965     setSubBoard();
966   } else {
967     setSubBoard();
968   }
969 }
970
971 if (rows % 3 === 1) {
972   if (rows / 3 % 3 === 0) {
973     setSubBoard();
974   } else {
975     setSubBoard();
976   }
977 }
978
979 if (rows % 3 === 2) {
980   if (rows / 3 % 3 === 0) {
981     setSubBoard();
982   } else {
983     setSubBoard();
984   }
985 }
986
987 if (rows % 3 === 0) {
988   if (rows / 3 % 3 === 0) {
989     setSubBoard();
990   } else {
991     setSubBoard();
992   }
993 }
994
995 if (rows % 3 === 1) {
996   if (rows / 3 % 3 === 0) {
997     setSubBoard();
998   } else {
999     setSubBoard();
1000   }
1001 }
1002
1003 if (rows % 3 === 2) {
1004   if (rows / 3 % 3 === 0) {
1005     setSubBoard();
1006   } else {
1007     setSubBoard();
1008   }
1009 }
1010
1011 if (rows % 3 === 0) {
1012   if (rows / 3 % 3 === 0) {
1013     setSubBoard();
1014   } else {
1015     setSubBoard();
1016   }
1017 }
1018
1019 if (rows % 3 === 1) {
1020   if (rows / 3 % 3 === 0) {
1021     setSubBoard();
1022   } else {
1023     setSubBoard();
1024   }
1025 }
1026
1027 if (rows % 3 === 2) {
1028   if (rows / 3 % 3 === 0) {
1029     setSubBoard();
1030   } else {
1031     setSubBoard();
1032   }
1033 }
1034
1035 if (rows % 3 === 0) {
1036   if (rows / 3 % 3 === 0) {
1037     setSubBoard();
1038   } else {
1039     setSubBoard();
1040   }
1041 }
1042
1043 if (rows % 3 === 1) {
1044   if (rows / 3 % 3 === 0) {
1045     setSubBoard();
1046   } else {
1047     setSubBoard();
1048   }
1049 }
1050
1051 if (rows % 3 === 2) {
1052   if (rows / 3 % 3 === 0) {
1053     setSubBoard();
1054   } else {
1055     setSubBoard();
1056   }
1057 }
1058
1059 if (rows % 3 === 0) {
1060   if (rows / 3 % 3 === 0) {
1061     setSubBoard();
1062   } else {
1063     setSubBoard();
1064   }
1065 }
1066
1067 if (rows % 3 === 1) {
1068   if (rows / 3 % 3 === 0) {
1069     setSubBoard();
1070   } else {
1071     setSubBoard();
1072   }
1073 }
1074
1075 if (rows % 3 === 2) {
1076   if (rows / 3 % 3 === 0) {
1077     setSubBoard();
1078   } else {
1079     setSubBoard();
1080   }
1081 }
1082
1083 if (rows % 3 === 0) {
1084   if (rows / 3 % 3 === 0) {
1085     setSubBoard();
1086   } else {
1087     setSubBoard();
1088   }
1089 }
1090
1091 if (rows % 3 === 1) {
1092   if (rows / 3 % 3 === 0) {
1093     setSubBoard();
1094   } else {
1095     setSubBoard();
1096   }
1097 }
1098
1099 if (rows % 3 === 2) {
1100   if (rows / 3 % 3 === 0) {
1101     setSubBoard();
1102   } else {
1103     setSubBoard();
1104   }
1105 }
1106
1107 if (rows % 3 === 0) {
1108   if (rows / 3 % 3 === 0) {
1109     setSubBoard();
1110   } else {
1111     setSubBoard();
1112   }
1113 }
1114
1115 if (rows % 3 === 1) {
1116   if (rows / 3 % 3 === 0) {
1117     setSubBoard();
1118   } else {
1119     setSubBoard();
1120   }
1121 }
1122
1123 if (rows % 3 === 2) {
1124   if (rows / 3 % 3 === 0) {
1125     setSubBoard();
1126   } else {
1127     setSubBoard();
1128   }
1129 }
1130
1131 if (rows % 3 === 0) {
1132   if (rows / 3 % 3 === 0) {
1133     setSubBoard();
1134   } else {
1135     setSubBoard();
1136   }
1137 }
1138
1139 if (rows % 3 === 1) {
1140   if (rows / 3 % 3 === 0) {
1141     setSubBoard();
1142   } else {
1143     setSubBoard();
1144   }
1145 }
1146
1147 if (rows % 3 === 2) {
1148   if (rows / 3 % 3 === 0) {
1149     setSubBoard();
1150   } else {
1151     setSubBoard();
1152   }
1153 }
1154
1155 if (rows % 3 === 0) {
1156   if (rows / 3 % 3 === 0) {
1157     setSubBoard();
1158   } else {
1159     setSubBoard();
1160   }
1161 }
1162
1163 if (rows % 3 === 1) {
1164   if (rows / 3 % 3 === 0) {
1165     setSubBoard();
1166   } else {
1167     setSubBoard();
1168   }
1169 }
1170
1171 if (rows % 3 === 2) {
1172   if (rows / 3 % 3 === 0) {
1173     setSubBoard();
1174   } else {
1175     setSubBoard();
1176   }
1177 }
1178
1179 if (rows % 3 === 0) {
1180   if (rows / 3 % 3 === 0) {
1181     setSubBoard();
1182   } else {
1183     setSubBoard();
1184   }
1185 }
1186
1187 if (rows % 3 === 1) {
1188   if (rows / 3 % 3 === 0) {
1189     setSubBoard();
1190   } else {
1191     setSubBoard();
1192   }
1193 }
1194
1195 if (rows % 3 === 2) {
1196   if (rows / 3 % 3 === 0) {
1197     setSubBoard();
1198   } else {
1199     setSubBoard();
1200   }
1201 }
1202
1203 if (rows % 3 === 0) {
1204   if (rows / 3 % 3 === 0) {
1205     setSubBoard();
1206   } else {
1207     setSubBoard();
1208   }
1209 }
1210
1211 if (rows % 3 === 1) {
1212   if (rows / 3 % 3 === 0) {
1213     setSubBoard();
1214   } else {
1215     setSubBoard();
1216   }
1217 }
1218
1219 if (rows % 3 === 2) {
1220   if (rows / 3 % 3 === 0) {
1221     setSubBoard();
1222   } else {
1223     setSubBoard();
1224   }
1225 }
1226
1227 if (rows % 3 === 0) {
1228   if (rows / 3 % 3 === 0) {
1229     setSubBoard();
1230   } else {
1231     setSubBoard();
1232   }
1233 }
1234
1235 if (rows % 3 === 1) {
1236   if (rows / 3 % 3 === 0) {
1237     setSubBoard();
1238   } else {
1239     setSubBoard();
1240   }
1241 }
1242
1243 if (rows % 3 === 2) {
1244   if (rows / 3 % 3 === 0) {
1245     setSubBoard();
1246   } else {
1247     setSubBoard();
1248   }
1249 }
1250
1251 if (rows % 3 === 0) {
1252   if (rows / 3 % 3 === 0) {
1253     setSubBoard();
1254   } else {
1255     setSubBoard();
1256   }
1257 }
1258
1259 if (rows % 3 === 1) {
1260   if (rows / 3 % 3 === 0) {
1261     setSubBoard();
1262   } else {
1263     setSubBoard();
1264   }
1265 }
1266
1267 if (rows % 3 === 2) {
1268   if (rows / 3 % 3 === 0) {
1269     setSubBoard();
1270   } else {
1271     setSubBoard();
1272   }
1273 }
1274
1275 if (rows % 3 === 0) {
1276   if (rows / 3 % 3 === 0) {
1277     setSubBoard();
1278   } else {
1279     setSubBoard();
1280   }
1281 }
1282
1283 if (rows % 3 === 1) {
1284   if (rows / 3 % 3 === 0) {
1285     setSubBoard();
1286   } else {
1287     setSubBoard();
1288   }
1289 }
1290
1291 if (rows % 3 === 2) {
1292   if (rows / 
```

# BAB V

## PENGUJIAN DAN HASIL ANALISIS

### 5.1 Hasil Pengujian

#### a. Test case 1

Test Case a.txt	
6 6 11 AAB..F .BCDF GPPCDFK GH.III GHJ... LLJMM.	
Algoritma	Output
UCS	<p>Solution</p> <p>Solution Length: 5 moves</p> <p>Nodes Visited: 200</p> <p>Execution Time: 58.40 ms</p> <p>Initial Board</p> <p>PREVIOUS 0 / 5 Next Auto Play</p> <p>Save Solution to File</p>
GBFS dengan heuristik Manhattan Distances	<p>Solution</p> <p>Solution Length: 21 moves</p> <p>Nodes Visited: 51</p> <p>Execution Time: 22.10 ms</p> <p>Initial Board</p> <p>PREVIOUS 0 / 21 Next Auto Play</p> <p>Save Solution to File</p>

### GBFS dengan heuristik Blocking Cells



### GBFS dengan heuristik Blocking Vehicles



### A\* dengan heuristik Manhattan Distances



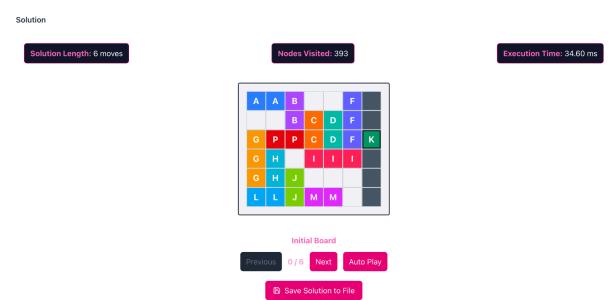
### A\* dengan heuristik Blocking Cells



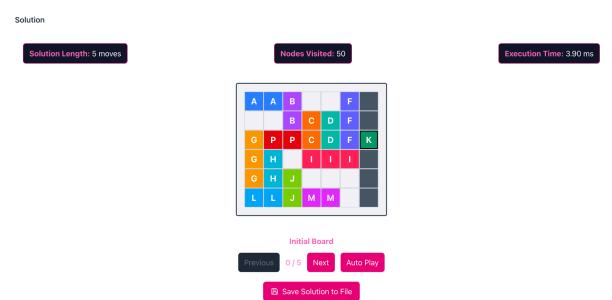
### A\* dengan heuristik Blocking Vehicles



### Fringe dengan heuristik Manhattan Distances



### Fringe dengan heuristik Blocking Cells



### Fringe dengan heuristik Blocking Vehicles



### b. Test case 2

#### Test Case b.txt

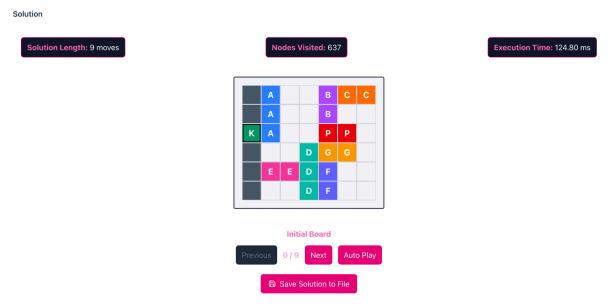
```
6 6
7
A..BCC
A..B..
KA..PP.
..DGG.
EEDF..
..DF..
```

Algoritma

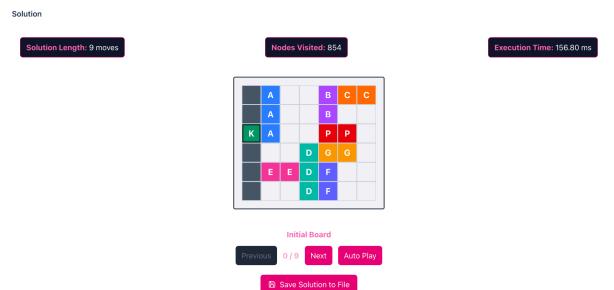
Output

UCS	<p>Solution</p> <p>Solution Length: 9 moves</p> <p>Nodes Visited: 1129</p> <p>Execution Time: 244.60 ms</p> <p>Initial Board</p> <p>Previous 0 / 9 Next Auto Play</p> <p>Save Solution to File</p>
GBFS dengan heuristik Manhattan Distances	<p>Solution</p> <p>Solution Length: 15 moves</p> <p>Nodes Visited: 267</p> <p>Execution Time: 38.20 ms</p> <p>Initial Board</p> <p>Previous 0 / 15 Next Auto Play</p> <p>Save Solution to File</p>
GBFS dengan heuristik Blocking Cells	<p>Solution</p> <p>Solution Length: 20 moves</p> <p>Nodes Visited: 226</p> <p>Execution Time: 51.00 ms</p> <p>Initial Board</p> <p>Previous 0 / 20 Next Auto Play</p> <p>Save Solution to File</p>
GBFS dengan heuristik Blocking Vehicles	<p>Solution</p> <p>Solution Length: 15 moves</p> <p>Nodes Visited: 159</p> <p>Execution Time: 36.60 ms</p> <p>Initial Board</p> <p>Previous 0 / 15 Next Auto Play</p> <p>Save Solution to File</p>
A* dengan heuristik Manhattan Distances	<p>Solution</p> <p>Solution Length: 10 moves</p> <p>Nodes Visited: 648</p> <p>Execution Time: 116.80 ms</p> <p>Initial Board</p> <p>Previous 0 / 10 Next Auto Play</p> <p>Save Solution to File</p>

### A\* dengan heuristik Blocking Cells



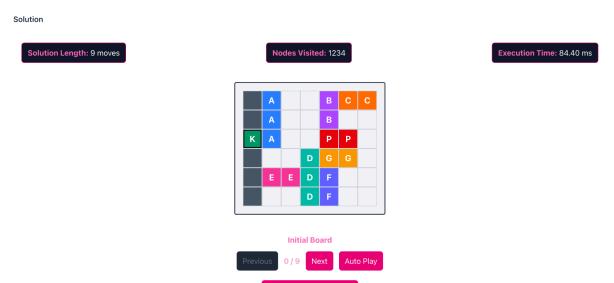
### A\* dengan heuristik Blocking Vehicles



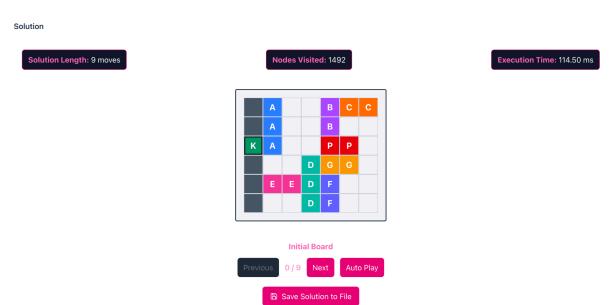
### Fringe dengan heuristik Manhattan Distances



### Fringe dengan heuristik Blocking Cells



### Fringe dengan heuristik Blocking Vehicles



### c. Test case 3

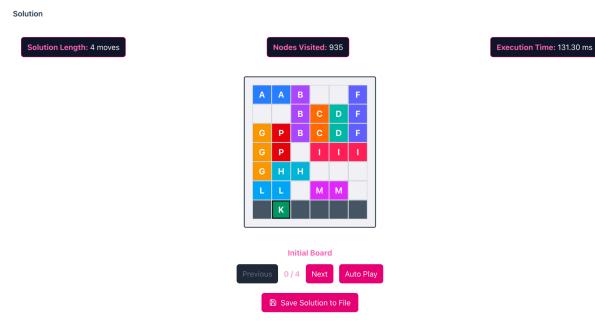
#### Test Case c.txt

```
6 6
10
AAB..F
..BCDF
GPBCDF
GP.III
GHH...
LL.MM.
K
```

#### Algoritma

UCS

#### Output



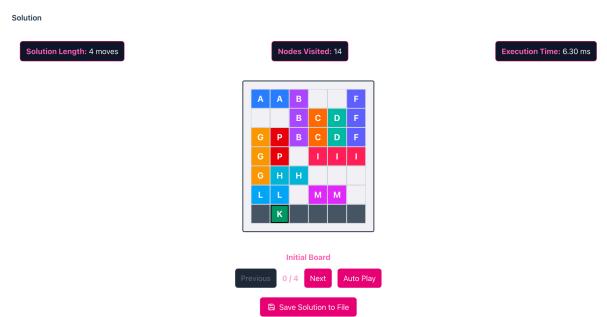
#### GBFS dengan heuristik Manhattan Distances



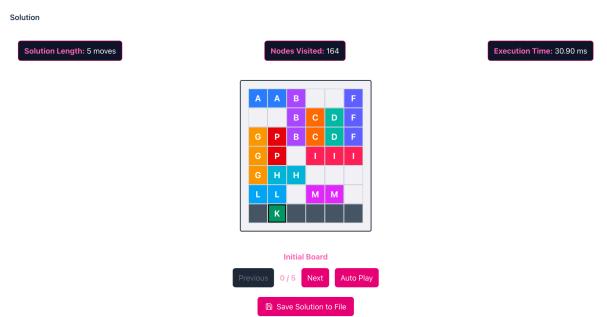
#### GBFS dengan heuristik Blocking Cells



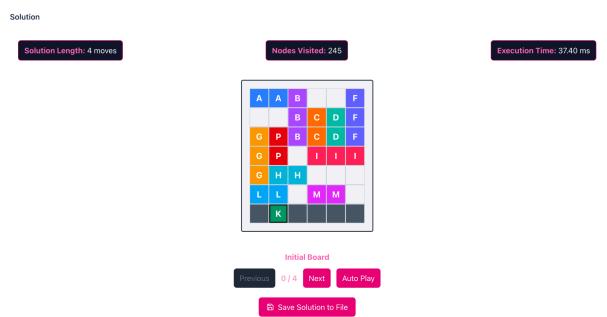
### GBFS dengan heuristik Blocking Vehicles



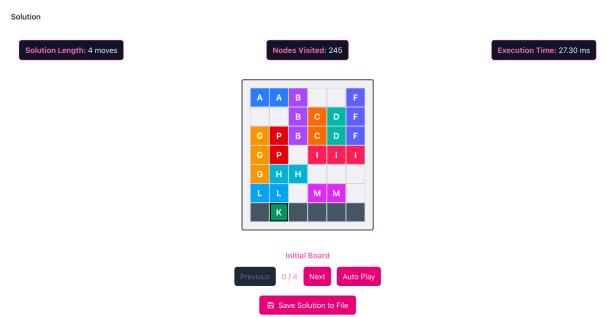
### A\* dengan heuristik Manhattan Distances



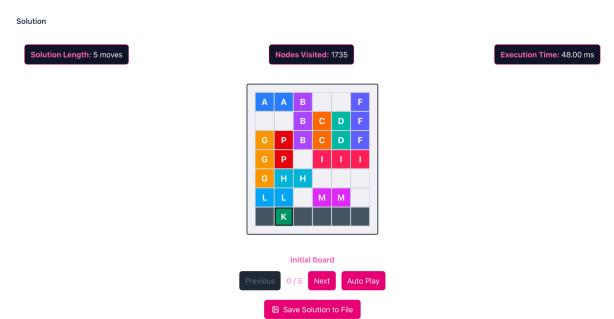
### A\* dengan heuristik Blocking Cells



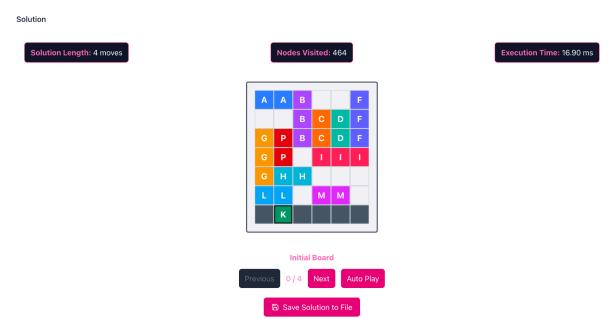
### A\* dengan heuristik Blocking Vehicles



### Fringe dengan heuristik Manhattan Distances



### Fringe dengan heuristik Blocking Cells



### Fringe dengan heuristik Blocking Vehicles



#### d. Test case 4

##### Test Case d.txt

```

6 6
10
K
AAB..F
..BCDF
GPBCDF
GP.III
GHH...
LL.MM.
  
```

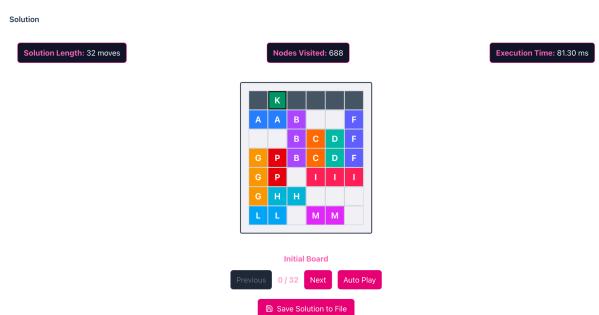
##### Algoritma

##### Output

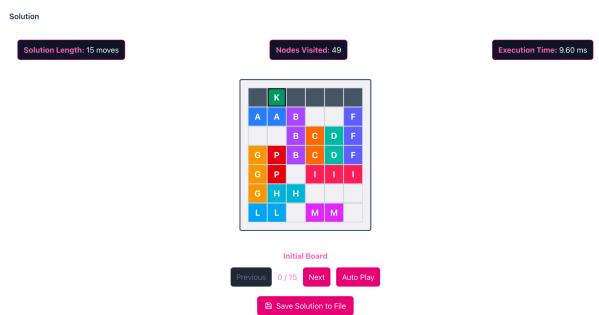
## UCS



## GBFS dengan heuristik Manhattan Distances



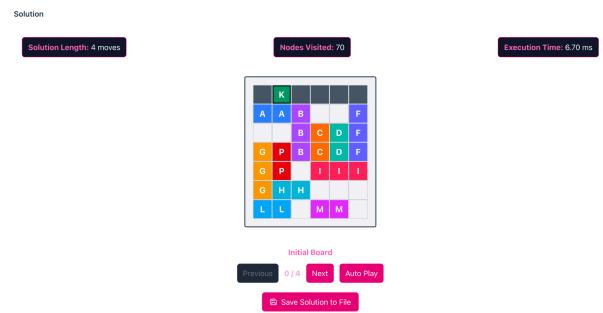
## GBFS dengan heuristik Blocking Cells



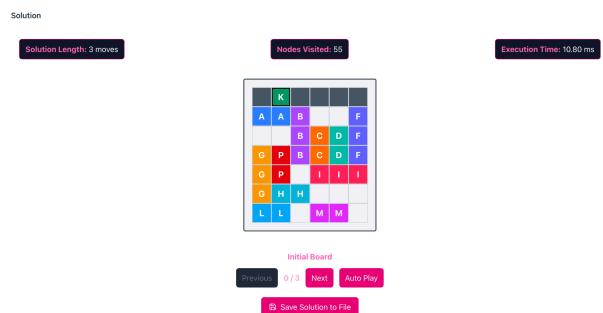
## GBFS dengan heuristik Blocking Vehicles



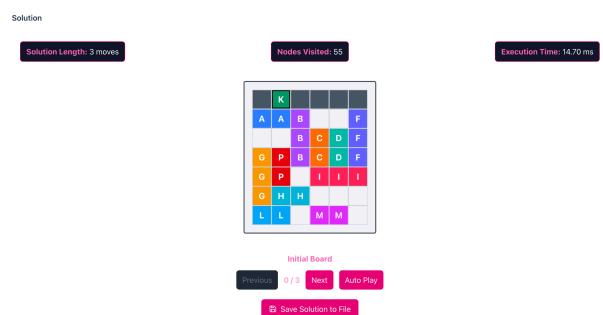
### A\* dengan heuristik Manhattan Distances



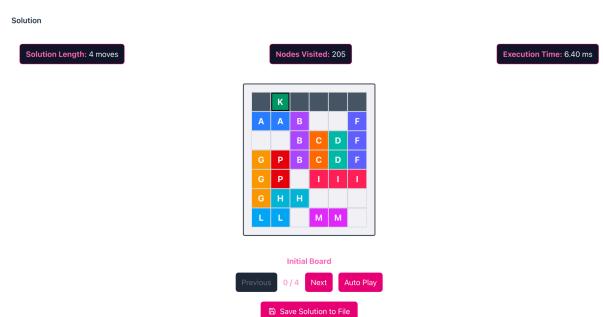
### A\* dengan heuristik Blocking Cells



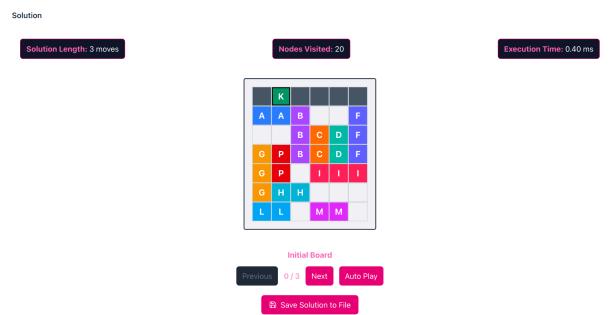
### A\* dengan heuristik Blocking Vehicles



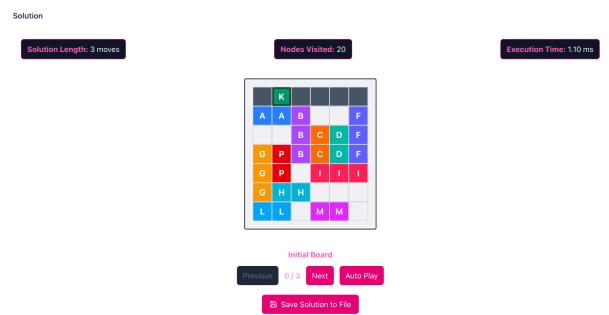
### Fringe dengan heuristik Manhattan Distances



### Fringe dengan heuristik Blocking Cells



### Fringe dengan heuristik Blocking Vehicles



#### e. Test case 5

##### Test Case e.txt

```
6 6
12
..ABCC
DDAB.E
KGFAPPE
GF.HH.
IJLLL
IJMM..
```

##### Algoritma

UCS

##### Output



### GBFS dengan heuristik Manhattan Distances



### GBFS dengan heuristik Blocking Cells



### GBFS dengan heuristik Blocking Vehicles



### A\* dengan heuristik Manhattan Distances



### A\* dengan heuristik Blocking Cells



### A\* dengan heuristik Blocking Vehicles



### Fringe dengan heuristik Manhattan Distances



### Fringe dengan heuristik Blocking Cells



### Fringe dengan heuristik Blocking Vehicles



### f. Test case 6

#### Test Case f.txt

```
6 6
11
AAB..F
..BCDFK
GPPCDF
GH.III
GHJ...
LLJMM.
```

Algoritma	Output																																				
Input tidak valid	<p>Puzzle cannot be solved: Primary piece is not aligned with the exit</p> <p>Puzzle Configuration</p> <p>The initial state of the puzzle board is as follows:</p> <table border="1"><tr><td>A</td><td>A</td><td>B</td><td></td><td></td><td>F</td></tr><tr><td></td><td>B</td><td>C</td><td>D</td><td>F</td><td>K</td></tr><tr><td>G</td><td>P</td><td>P</td><td>C</td><td>D</td><td>F</td></tr><tr><td>G</td><td>H</td><td>I</td><td>I</td><td>I</td><td></td></tr><tr><td>G</td><td>H</td><td>J</td><td></td><td></td><td></td></tr><tr><td>L</td><td>L</td><td>J</td><td>M</td><td>M</td><td></td></tr></table>	A	A	B			F		B	C	D	F	K	G	P	P	C	D	F	G	H	I	I	I		G	H	J				L	L	J	M	M	
A	A	B			F																																
	B	C	D	F	K																																
G	P	P	C	D	F																																
G	H	I	I	I																																	
G	H	J																																			
L	L	J	M	M																																	

### g. Test case 7

#### Test Case g.txt

```
4 4
0
..PPK
....
```

Algoritma	Output
-----------	--------

Input tidak valid	<input type="button" value="Upload Puzzle File"/> Selected: g.txt  Failed to parse file: Invalid file format: Expected at least 4 rows for the board configuration
-------------------	---

#### h. Test case 8

Test Case h.txt	
Algoritma	Output
Input tidak valid	<input type="button" value="Upload Puzzle File"/> Selected: h.txt  Failed to parse file: Invalid file format: Missing required configuration data

## 5.2 Analisis Pengujian

### 5.2.1 Algoritma Uniform Cost Search (UCS)

UCS mengekspansi setiap simpul sampai biaya kumulatifnya mencapai biaya solusi optimal  $C$ . Karena tiap kenaikan biaya minimal sebesar  $\epsilon$  memicu eksplorasi cabang baru, jumlah level yang dieksplorasi adalah  $1 + C/\epsilon$ . Di setiap level, ada hingga  $b$  cabang, sehingga totalnya  $b^{(1 + C/\epsilon)}$ .

Jika setiap langkah memiliki biaya sama, misalnya pada permainan Rush Hour dimana setiap langkah memiliki biaya = 1, maka  $C = d$  dan  $\epsilon = 1$ , sehingga kompleksitasnya menjadi  $O(b^d)$  yang mirip dengan BFS.

### 5.2.2 Algoritma Greedy Best First Search (GBFS)

GBFS hanya mengejar heuristik  $h(n)$  tanpa memperhitungkan  $g(n)$ . Dalam skenario terburuk (heuristik sama sekali tidak informatif), algoritma bisa mengeksplorasi seluruh pohon sampai kedalaman maksimum  $m$ . Karena tiap simpul membuka rata-rata  $bbb$  anak, total simpul yang diproses hingga kedalaman  $m$  adalah  $1 + b + b^2 + \dots + b^m = O(b^m)$ .

### 5.2.3 Algoritma A\*

A\* mengekspansi semua simpul dengan  $f(n) = g(n) + h(n)$  (dengan heuristik yang admissible). Jumlah simpul ini pada level hingga kedalaman  $d$  tumbuh eksponensial, sehingga kompleksitas waktu nya  $O(b^d)$ . Kinerja riil sangat bergantung pada kualitas heuristik. Semakin mendekati biaya sebenarnya, makin sedikit simpul yang perlu dilihat.

#### **5.2.4 Algoritma Fringe Search**

Mirip dengan algoritma A\*, Fringe Search melakukan iterative-deepening berdasarkan ambang nilai  $f$ . Setiap iterasi dapat memproses sebagian fringe hingga kedalaman solusi  $d$ . Dalam kasus terburuk, total simpul yang diproses masih  $O(b^d)$ , meski overhead iterasi tambahan biasanya kecil dibandingkan IDA\*.

## **BAB VI**

### **PENJELASAN BONUS**

#### **6.1 Algoritma Fringe Search**

Fringe Search merupakan varian A\* yang dirancang menekan penggunaan memori. Fringe Search diterapkan dengan menggunakan empat variabel utama, yaitu fringe, cache, threshold, dan nextThreshold. Pada langkah awal, initialState dibuat dari konfigurasi papan permainan, lalu dihitung biaya dari awal ( $g(n)$ ) dan nilai heuristik, sehingga  $f(n) = g(n) + \text{heuristik}$ . Semua status yang akan diperiksa dikumpulkan dalam daftar fringe. Status pertama diambil dan diperiksa apakah sudah menyelesaikan puzzle. Jika ya, jalur solusi dikembalikan. Apabila nilai  $f(n)$  status tersebut melebihi batas (threshold) saat ini, status ditunda dan nilai  $f(n)$  terkecil yang melampaui batas dicatat sebagai nextThreshold untuk iterasi selanjutnya. Jika  $f(n)$  masih dalam batas, status dikembangkan menjadi semua penerusnya, di mana setiap penerus dihitung ulang  $g(n)$  dan heuristik nya, kemudian disaring dengan cache untuk menghindari jalur yang lebih mahal. Penerus yang lolos seleksi diurutkan kembali berdasarkan nilai  $f(n)$  dan dimasukkan kembali ke fringe. Setelah satu siklus habis, apabila tidak ada status baru di bawah threshold, batas dinaikkan menjadi nextThreshold dan proses diulang hingga solusi ditemukan atau fringe kosong. Implementasi ini menjamin efisiensi memori karena hanya menyimpan node-node dalam batas threshold tertentu.

#### **6.2 Fungsi Heuristik**

Terdapat beberapa fungsi heuristik yang dapat digunakan untuk pencarian algoritma GBFS, A\*, dan Fringe Search, yaitu sebagai berikut.

##### **6.2.1 Manhattan Distance Heuristic**

Mengukur jarak absolut antara batas paling ujung potongan utama dan posisi pintu keluar pada arah orientasi potongan (horizontal atau vertikal).

##### **6.2.2 Blocking Vehicles Heuristic**

Menghitung jumlah kendaraan penghalang di jalur menuju pintu keluar. Setiap kendaraan yang menutup jalur utama dihitung satu kali, dengan mekanisme loncatan melewati seluruh panjang kendaraan.

### **6.2.3 Blocking Cells Heuristic**

Menghitung jumlah kotak terisi (terlepas dari apakah merupakan potongan utama atau bukan) sepanjang jalur menuju pintu keluar.

## **6.3 Graphical User Interface**

Antarmuka pengguna dikembangkan menggunakan React dan Vite sebagai framework dengan TailwindCSS untuk styling. Komponen utama meliputi:

- Home, yang menampilkan judul aplikasi dan informasi singkat.
- About, yang menampilkan informasi program serta pembuatnya.
- Input Section, yang menerima konfigurasi papan permainan dari pengguna.
- Board Display, yang menampilkan tata letak papan permainan secara visual.
- Control Panel, yang Mengatur pilihan algoritma (UCS, A\*, Greedy BFS, Dijkstra, Fringe Search) beserta fungsi heuristik (jika tersedia) melalui menu dropdown.
- Solution Display, yang menampilkan metrik kinerja, seperti jumlah node yang dikunjungi, waktu eksekusi, dan visualisasi pencarian path secara real-time.

## **BAB VII**

### **LAMPIRAN**

#### **7.1 Daftar Pustaka**

Maulidevi, Nur Ulfa 2025. "Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search"

([https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf), diakses 16 Mei 2025).

Munir, Rinaldi dan Maulidevi, N. U. 2025. "Penentuan Rute (Route/Path Planning) Bagian 2: Algoritma A\*"

([https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf), diakses 16 Mei 2025).

#### **7.2 Tautan Repository**

Link repository dari Tugas Kecil 3 IF2211 Strategi Algoritma adalah sebagai berikut.

[https://github.com/Incheon21/Tucil3\\_13523029\\_13523033](https://github.com/Incheon21/Tucil3_13523029_13523033)

#### **7.3 Tautan Deployment**

Link website dari Tugas Kecil 3 IF2211 Strategi Algoritma adalah sebagai berikut.

<https://tucil3-13523029-13523033-e3g6.vercel.app/>

#### **7.4 Tabel Evaluasi**

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4. Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	✓	
5. [Bonus] Implementasi algoritma pathfinding alternatif	✓	

6. [Bonus] Implementasi 2 atau lebih heuristik alternatif	✓	
7. [Bonus] Program memiliki GUI	✓	
8. Program dan laporan dibuat (kelompok) sendiri	✓	