*Penyelesaian Rush Hour Puzzle Menggunakan Algoritma A**

Tony Eko Yuwono 13518030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
tony.yuwono@gmail.com

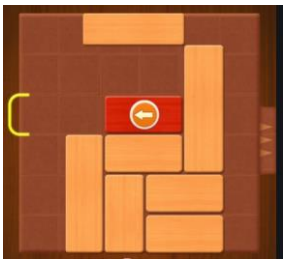
***Abstraksi*—Masalah penentuan rute terpendek merupakan masalah yang sering dibicarakan di dunia informatika. Masalah penentuan rute yang biasa dibicarakan merupakan masalah pencarian rute terpendek dari satu simpul awal menuju satu simpul tujuan. Salah satu *logic puzzle* yang menggunakan konsep penentuan rute terpendek adalah *Rush Hour*, sebuah puzzle dimana pemain harus menggeser mobil-mobil penghalang yang ada agar mobil pemain dapat keluar dari papan *puzzle*. Pada makalah ini, akan dijelaskan penggunaan algoritma A* untuk menyelesaikan *puzzle Rush Hour*.**

Keywords—Algoritma A*, Jalur Terpendek, Puzzle, Rush Hour

I. PENDAHULUAN

Penentuan rute terpendek adalah salah satu masalah yang sering dibicarakan di dunia informatika karena masalah ini memiliki banyak korespondensi dengan masalah yang ada di dunia nyata. Ada banyak algoritma yang dapat menyelesaikan masalah penentuan rute terpendek ini, mulai dari algoritma *Brute Force*, Dijkstra, *Uniform Cost Search* (UCS), *Greedy Best-First Search*, dan algoritma A*. Salah satu algoritma paling mangkus untuk penyelesaian masalah ini adalah algoritma A* (*A Star*). Algoritma ini dianggap mangkus karena pendekatan algoritma A* merupakan gabungan dari algoritma *Uniform Cost Search* (UCS) dan algoritma *Greedy Best-First Search*. Dengan menggunakan algoritma A*, kita dapat mencari sebuah jalur terpendek dari suatu simpul awal ke simpul tujuan dengan lebih efektif dan mangkus.

Salah satu permainan *logic puzzle* yang memanfaatkan konsep penentuan rute terpendek adalah permainan klasik *Rush Hour* yang ditemukan tahun 1970-an dan pada awalnya merupakan permainan puzzle berbentuk fisik yang terdiri dari sebuah papan dan mobil-mobil penghalang. Namun, saat ini permainan ini dapat kita temukan pada *smartphone* kita (baik iOS maupun Android). Ada banyak sekali versi permainan yang dikembangkan oleh berbagai macam pengembang yang mengadaptasi permainan *Rush Hour* ini. Permainan *Rush Hour* merupakan permainan dimana pemain harus mengeluarkan mobil pemain dari sebuah kemacetan antarmobil sehingga menyebabkan mobil pemain tidak dapat bergerak maju meninggalkan papan permainan. Karena permainan ini menerapkan konsep penentuan rute, maka permainan ini dapat diselesaikan menggunakan algoritma yang berguna untuk menentukan rute seperti algoritma A*. Pada makalah ini, akan dibahas penyelesaian permainan *Rush Hour* menggunakan algoritma A*. Berikut adalah contoh gambar permainan ini.



Gambar 1. Contoh *Puzzle Rush Hour*
Sumber: http://www.appsting.com/apps/731297925-unblock-it-free-traffic-rush-hour-block-App

II. DASAR TEORI

A. *Graf*

Graf adalah suatu representasi visual yang terdiri dari simpul (*vertex)* dan sisi (*edge*). Secara umum, graf didefinisikan:

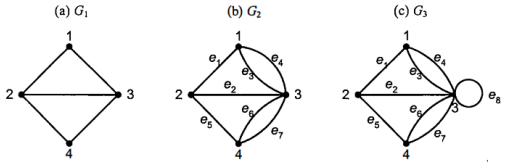
$$G = (V, E)$$

dengan V adalah himpunan tidak kosong dari simpul-simpul ($V \neq \emptyset$), dan E adalah himpunan sisi yang menghubungkan sepasang simpul (boleh himpunan kosong) [3].

Graf dibagi menjadi dua jenis berdasarkan adanya sisi ganda atau gelang (sisi yang menghubungkan sebuah simpul yang sama):

1. Graf sederhana (*simple graph*)
   Graf sederhana adalah graf yang tidak mengandung sisi ganda maupun gelang. Gambar 2(a) adalah contoh graf sederhana. Pada graf sederhana sisi (*a*, *b*) = (*b*, *a*), seperti contoh pada Gambar 2(a) sisi (1,2) sama dengan (2,1).

2. Graf tak-sederhana (*unsimple graph*)
   Graf tak-sederhana adalah graf yang mengandung sisi ganda atau gelang (*loop*). Graf yang mengandung sisi ganda dinamakan graf ganda, sedangkan graf yang memiliki sisi gelang (*loop*) dinamakan graf semu. Gambar 1 (b) dan (c) adalah contoh graf ganda dan graf semu.

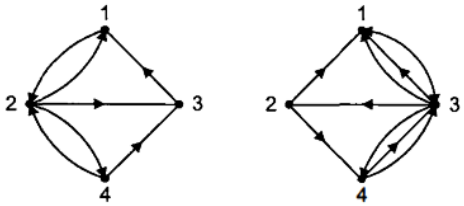


Gambar 2. (a) Graf sederhana, (b) graf ganda, (c) graf semu
Sumber: Matematika Diskrit Edisi 3, Rinaldi Munir, Halaman 356

Selain itu, berdasarkan orientasi arah, graf dibedakan menjadi dua jenis, yaitu:

1. Graf tak-berarah (*undirected graph*)
   Graf tak berarah adalah graf yang tidak mempunyai orientasi arah sehingga urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Misalkan terdapat sisi (*u, v*) pada graf *G*, maka sisi (*v, u*) adalah sisi yang sama. Semua graf pada Gambar 2 merupakan graf tak-berarah.

2. Graf berarah (*directed graph/digraph*)
   Graf berarah adalah graf yang setiap sisinya diberikan orientasi arah. Misalkan terdapat sisi (*u, v*) pada graf *G*, maka sisi (*v, u*) adalah sisi yang berbeda, dengan kata lain (*u, v*) ≠ (*v, u*). Untuk sisi (*u, v*), simpul *u* dinamakan simpul asal dan simpul *v* dinamakan simpul terminal. Semua graf pada Gambar 3 merupakan graf berarah.



Gambar 3. (a) Graf berarah, (b) graf-ganda berarah
Sumber: Matematika Diskrit Edisi 3, Rinaldi Munir, Halaman 359

B. *Path Finding*

Path Finding (penentuan/pencarian jalur) adalah proses penentuan jalur dari suatu simpul awal ke simpul tujuan. Masalah penentuan jalur biasanya dapat kita temukan pada persoalan graf dan penentuan jalur yang ingin dicari adalah penentuan jalur terpendek dari suatu simpul awal ke simpul tujuan.

C. *Shortest Path*

Shortest Path (jalur terpendek) merupakan masalah optimalisasi yang sering dicari pada persoalan *path finding*. Terdapat banyak algoritma yang dapat menyelesaikan masalah penentuan rute terpendek ini, mulai dari algoritma *Brute Force*, Dijkstra, *Uniform Cost Search* (UCS), *Greedy Best-First Search*, dan algoritma A*.

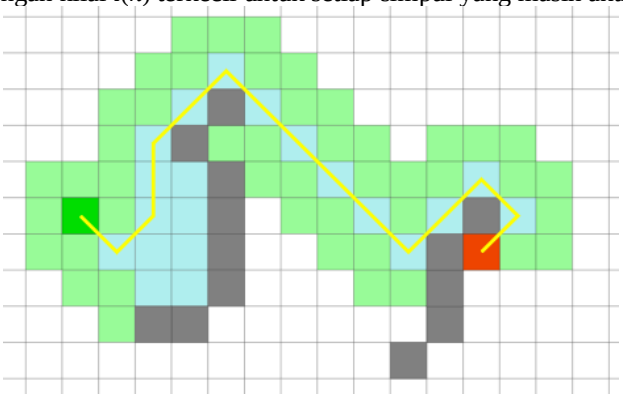
D. *Algoritma A**

Algoritma A* adalah algoritma yang banyak digunakan untuk masalah pencarian jalur terutama masalah pencarian jalur terpendek di dalam suatu graf. Selain itu, algoritma ini juga dapat digunakan untuk mencari jalur terdekat dari suati simpul awal ke simpul tujuan jika terdapat suatu lokasi yang tidak dapat dilewati pada saat melakukan penelusuran. Algoritma ini ditemukan oleh Peter Hard, Nils Nilsson, dan Bertram Raphael pada tahun 1968. Algoritma ini termasuk dalam algoritma pencarian jalur bertipe *informed (heuristic) search.* Pencarian bertipe *informed search* ini menggunakan penerapan nilai heuristik kepada permasalahan yang dihadapi disamping dari definisi masalah itu sendiri. Sehingga pencarian jalur bertipe *informed search* ini mampu menemukan solusi secara lebih efisien daripada pencarian jalur bertipe *uninformed search* seperti *Breadth First Search* (BFS) atau *Depth First Search* (DFS) [2].

Algoritma A* merupakan gabungan dari dua algorima pencarian jalur yaitu algoritma *Uniform Cost Search* (UCS) dan algoritma *Greedy Best-First Search*. Algoritma *Uniform Cost Search* (UCS) sendiri merupakan algoritma pencarian jalur bertipe *uninformed search* dengan penentuan jalur ekspansi pohon ruang status menggunakan fungsi $g(n)$ = panjang (harga) lintasan dari simpul asal ke simpul-n. Setiap langkah akan memilih nilai fungsi $g(n)$ terkecil dari simpul-simpul yang masih aktif. Sedangkan algoritma *Greedy Best-First Search* merupakan algoritma pencarian jalur bertipe *informed search* dengan fungsi greedy sebagai fungsi untuk penentuan jalur ekspansi pohon ruang status pada simpul-simpul yang aktif. Fungsi greedy ini dihitung menggunakan fungsi *heuristic* h(n) yang merupakan nilai taksiran ongkos dari simpul n ke simpul tujuan. Oleh karena itu fungsi yang digunakan oleh algoritma A* adalah

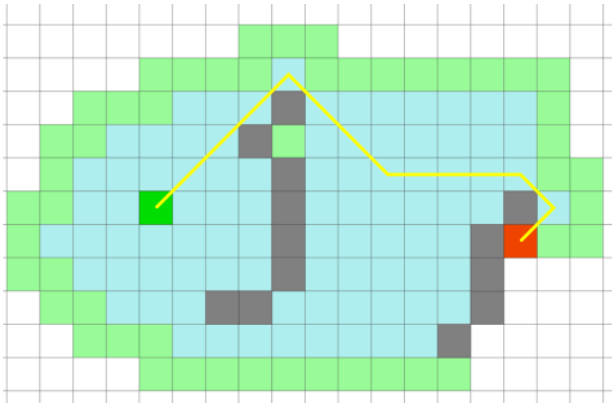
$$f(n) = g(n) + h(n)$$

Dimana $g(n)$ adalah fungsi untuk mencari panjang (harga) lintasan dari simpul asal ke simpul n dan h(n) adalah fungsi untuk menaksir ongkos dari simpul n ke simpul tujuan. Pada setiap langkahnya, algoritma A* akan membangkitkan simpul dengan nilai f(n) terkecil untuk setiap simpul yang masih aktif.



Gambar 4. Proses Pencarian Menggunakan *Greedy Best-First Search*
Sumber: https://qiao.github.io/PathFinding.js/visual/

Karena algoritma A* merupakan algoritma gabungan dari kedua algoritma tersebut, tentunya algoritma A* membutuhkan waktu yang lebih lama untuk menjelajahi semua kemungkinan jalur yang ada untuk mencari jalur solusi terpendek. Walaupun membutuhkan waktu yang lebih lama, algoritma A* dijamin akan menghasilkan solusi yang optimal. Salah satu contoh dapat dilihat pada Gambar 4 dan Gambar 5. Gambar 4 merupakan contoh penggunaan algoritma *Greedy Best-First Search* dan Gambar 5 merupakan contoh pengugnaan algoritma A* dalam pencarian jalur terpendek antara simpul awal ke simpul tujuan.



Gambar 5. Proses Pencarian Menggunakan Algoritma A*
Sumber: https://qiao.github.io/PathFinding.js/visual/

Pada Gambar 4, dapat dilihat bahwa proses pencarian menggunakan algoritma *Greedy Best-First Search* lebih cepat dibandingkan dengan algoritma A* pada Gambar 5, hal ini ditandai dengan jumlah simpul yang dibangkitkan pada Gambar 4 berjumlah lebih sedikit daripada jumlah simpul yang dibangkitkan pada Gambar 5. Namun, walaupun algoritma A* lebih lambat dibandingkan *Greedy Best-First Search,* algoritma A* selalu menghasilkan solusi yang optimum, sedangkan pada *Greedy Best-First Search* tidak selalu menghasilkan solusi yang optimum. Hal ini dapat dilihat dari jumlah simpul jalur solusi pada algoritma A* berjumlah 12 simpul (tidak termasuk simpul awal dan simpul tujuan) sedangkan pada algoritma *Greedy Best-First Search* menghasilkan simpul solusi sebanyak 14 simpul.

E. *Fungsi Heuristic*

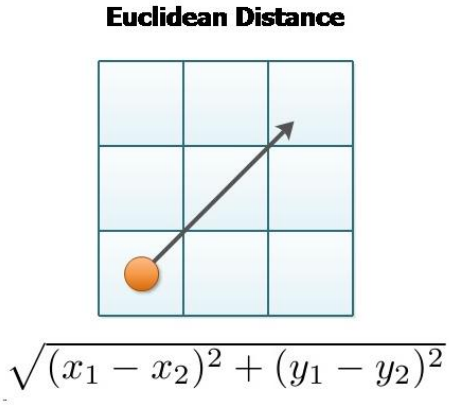
Algoritma A* menggunakan fungsi *heuristic* untuk menentukan *cost* demi mengambil keputusan pembangkitan jalur ekspansi yang akan dibangkitkan pada setiap langkahnya. Dalam permasalahan jarak, terdapat tiga variasi fungsi *heuristic* yang sering digunakan dalam algoritma A*, yaitu *Euclidean Distance*, *Manhattan Distance*, dan *Chebyshev Distance* [4].

1. *Euclidean Distance*

Euclidean Distance merupakan fungsi *heuristic* yang menghitung *cost* dengan cara menghitung jarak antara dua simpul dengan menggunakan rumus jarak antara dua titik, yaitu:

$$d = \sqrt{((X_1 - X_2)^2 + (Y_1 - Y_2)^2)}$$

dimana d adalan hasil jarak antara dua titik seperti pada Gambar 6.

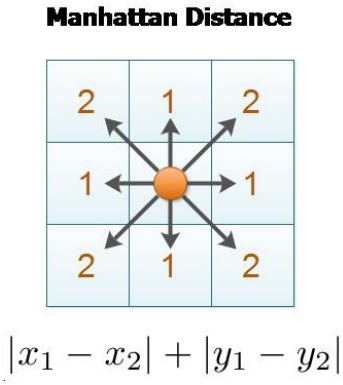


Gambar 6. *Euclidean Distance*
Sumber: https://iq.opengenus.org/euclidean-vs-manhattan-vs-chebyshev-distance/

2. *Manhattan Distance*

Manhattan Distance merupakan fungsi *heuristic* untuk menentukan *cost* dengan menghitung jarak dari simpul awal ke simpul tujuan dengan penjumlahan selisih jarak pada sumbu X dan selisih jarak pada sumbu Y.

$$d = | X_1 - X_2 | + | Y_1 - Y_2 |$$

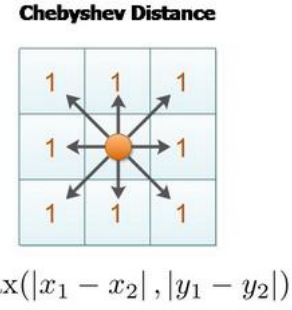


Gambar 7. *Euclidean Distance*
Sumber: https://iq.opengenus.org/euclidean-vs-manhattan-vs-chebyshev-distance/

3. *Chebyshev Distance*

Chebyshev Distance merupakan fungsi *heuristic* untuk menentukan *cost* dengan menghitung jarak dari suatu simpul awal ke simpul tujuan dengan menghitung nilai mutlak dari selisih pasangan koordinat simpul awal dan simpul tujuan.

d =$max[(X_1-X_2), (Y_1-Y_2)]$



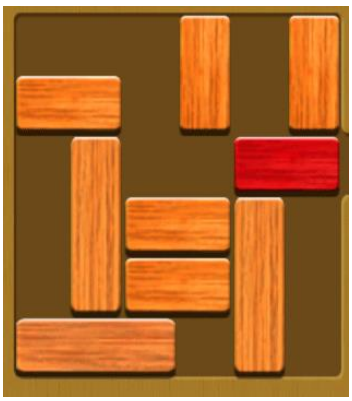
Gambar 8. *Chebyshev Distance*
Sumber: https://iq.opengenus.org/euclidean-vs-manhattan-vs-chebyshev-distance/

F. *Rush Hour*

Rush Hour adalah permainan berjenis *puzzle* yang terdapat pada perangkat berbasis Android atau iOS ataupun *puzzle* berbentuk fisik. *Rush Hour* sendiri awalnya ditemukan oleh Nob Yoshigahara pada tahun 1970-an. *Puzzle* ini mulai dikomersialkan pada tahun 1996 oleh perusahaan bernama *ThinkFun* (sebelumnya bernama *Binary Arts*). Papan permainan *puzzle* ini berukuran 6x6 dengan berbagai blok rintangan dengan ukuran yang berbeda-beda sehingga dapat dikombinasikan dalam susunan yang berbeda untuk membentuk blokade terhadap pintu keluar yang menjadi tujuan dari *puzzle* ini. Oleh karena itu tujuan dari permainan ini adalah memindahkan blok-blok *puzzle* yang dapat dipindahkan sedemikian sehingga blok *puzzle* yang berwarna merah dapat menjangkau pintu keluar yang sudah tersedia.

III. Definisi Persoalan

Sebelum menyelesaikan sebuah persoalan, kita harus mendefinisikan terlebih dahulu persoalan tersebut. Persoalan pada *puzzle Rush Hour* ini adalah mencari langkah untuk memindahkan blok-blok rintangan seminimal mungkin agar sebuah blok yang berwarna merah dapat berpindah menuju pintu keluar dari papan permainan *puzzle Rush Hour*. Berikut merupakan contoh *final state* pada *puzzle Rush Hour*.



Gambar 9. *Final state puzzle Rush Hour*
Sumber: https://www.newsinitiative.org/unblock-me-for-pc-windows-78xp/

IV. Penentuan Fungsi *Heuristic*

Untuk membuat sebuah algoritma A* untuk menyelesaikan *puzzle Rush Hour*, diperlukan sebuah fungsi *heuristic* yang menjadi salah satu ciri khas dari algoritma A*. Pemilihan fungsi *heuristic* sangat penting dalam algoritma A* karena fungsi *heuristic* tersebut menjadi salah satu penentu *cost* sebuah simpul pada pohon ruang status yang nantinya harus diekspansi. Seperti yang diketahui, terdapat tiga buah fungsi *heuristic* yang biasa digunakan dalam algoritma A*, yaitu: *Euclidean*, *Manhattan*, dan *Chebyshev Distance*.

Euclidean distance merupakan fungsi *heuristic* yang tepat untuk digunakan jika pergerakan dapat terjadi secara diagonal, karena fungsi *heuristic* ini menghitung jarak antara dua titik berdasarkan koordinat kedua titik tersebut dengan menggunakan rumus jarak antara dua titik. Pada *puzzle Rush Hour*, blok-blok hanya dapat bergerak secara horizontal atau vertikal. Oleh karena itu fungsi *heuristic* ini kurang cocok diterapkan untuk memecahkan *puzzle Rush Hour.*

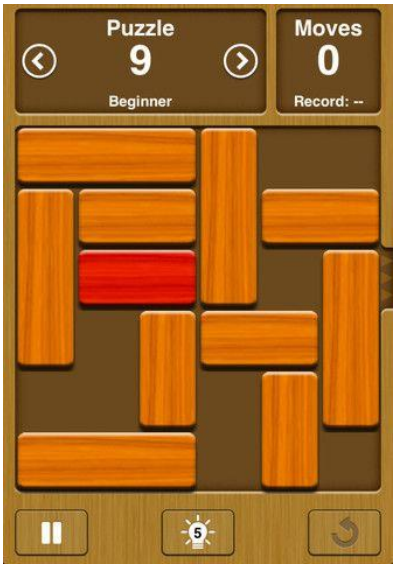
Manhattan distance merupakan fungsi *heuristic* yang tepat untuk digunakan jika pergerakan hanya terjadi secara horizontal ataupun vertikal saja, karena fungsi *heuristic* ini menghitung jarak antara dua titik berdasarkan banyaknya titik-titik minimal yang harus dilalui dalam koordinat x dan y. Oleh karena itu, fungsi *heuristic* ini dirasa cocok untuk menjadi fungsi *heuristic* pada pemecahan masalah *puzzle Rush Hour.*

Chebyshev distance merupakan fungsi *heuristic* yang tepat untuk digunakan jika pergerakan dapat terjadi secara diagonal seperti pada *euclidean distance*. Perbedaannya dengan *euclidean distance* yaitu pada *chebyshev distance cost* untuk langkah diagonal sama seperti *cost* untuk langkah horizontal maupun vertikal, sedangkan pada *euclidean distance* langkah yang dibutuhkan untuk langkah diagonal lebih besar daripada langkah horizontal maupun vertikal.

Berdasarkan analisa fungsi *heuristic* di atas, fungsi *heuristic* yang cocok untuk menyelesaikan masalah *puzzle Rush Hour* adalah fungsi *heuristic* berbasis *manhattan distance*. Ada dua buah fungsi *heuristic* berbasis *manhattan distance* yang akan digunakan untuk menyelesaikan *puzzle* ini, yaitu:

1. Jarak blok berwarna merah ke pintu keluar

   Nilai *heuristic* ini mengukur berapa jarak yang dibutuhkan oleh blok berwarna merah untuk mencapai pintu keluar. Nilai *heuristic* ini adalah nilai *heuristic* yang *admissible* karena untuk mencapai *goal state* menggunakan algoritma A*, jarak minimum yang harus ditempuh pastilah jarak blok berwarna merah ke pintu keluar. Semakin jauh jarak blok berwarna merah dari pintu keluar, maka pastilah blok merah tersebut semakin menjauhi pintu keluar.



Gambar 10. Ilustrasi untuk Fungsi *Heuristic*
Sumber:https://www.pinterest.com/pin/157555686934295536/

Dengan menggunakan fungsi *heuristic* yang menghitung jarak blok berwarna merah ke pintu keluar, diperoleh nilai *heuristic* untuk *state* pada Gambar 10 adalah berjumlah 3.

2. Banyak blok yang menghalangi pintu keluar

   Nilai *heuristic* ini mengukur berapa banyak blok yang menghalangi blok berwarna merah untuk mencapai pintu keluar. Nilai *heuristic* ini adalah nilai *heuristic* yang *admissible* karena untuk mencapai *goal state* menggunakan algoritma A*, jarak minimum yang harus ditempuh sebanding dengan banyak blok yang menghalangi pintu keluar. Semakin banyak blok yang menghalangi pintu keluar, maka blok berwarna merah akan semakin sulit mencapai pintu keluar. Dengan menggunakan fungsi *heuristic* yang menghitung banyak blok yang menghalangi pintu keluar, diperoleh nilai *heuristic* untuk *state* pada Gambar 9 adalah berjumlah 2.

Pada makalah ini, akan dibahas penyelesaian *puzzle Rush Hour* dengan menggunakan fungsi *heuristic* pertama, yaitu jarak blok berwarna merah ke pintu keluar.

V. Perancangan Algoritma

A. Pemetaan *Puzzle*

Untuk mendapatkan kondisi *puzzle* pada sebuah *state*, maka *puzzle* tersebut akan dipetakan menjadi matriks 2D. Berikut merupakan contoh tabel untuk *state puzzle* pada Gambar 10:

A	A	A	B	.	.
D	C	C	B	J	J
D	r	r	B	.	I
D	.	E	G	G	I
.	.	E	.	H	I
F	F	F	.	H	.

Tabel 1. Contoh Pemetaan *Puzzle*

Pada tabel 1, simbol r merupakan blok berwarna merah yang memiliki ukuran 2 blok. Selain itu terdapat blok rintangan A dan F yang merupakan blok rintangan horizontal yang berukuran 3 blok dan blok rintangan C, G, dan J yang merupakan blok rintangan horizontal yang berukuran 2 blok. Blok rintangan D, B, dan I merupakan blok rintangan vertikal yang berukuran 3 blok, sedangkan blok rintangan E dan H merupakan blok rintangan vertikal yang berukuran 2 blok. *Final state* dicapai jika blok r berada pada blok ke (5,3) dan (6,3) seperti pada tabel dibawah ini:

.	.	E	A	A	A
C	C	E	.	J	J
D	.	.	.	r	r
D	G	G	B	H	I
D	.	.	B	H	I
F	F	F	B	.	I

Tabel 2. Contoh *Final State Puzzle*

B. Algoritma untuk Pemecahan *Puzzle*

Setelah mengetahui *state* awal dari *puzzle Rush Hour* dan melakukan konversi menjadi matriks 2D, tahap berikutnya adalah membuat algoritma A* untuk menentukan pembangkitan simpul pada pohon ruang status untuk menyelesaikan *puzzle Rush Hour*.

Algoritma A* untuk menyelesaikan *puzzle Rush Hour* ini mirip dengan algoritma untuk menyelesaikan *15-Puzzle* yang sudah dikerjakan pada Tugas Kecil 3 Strategi Algoritma. Namun, jika pada *15-Puzzle* gerakan yang ditelusuri hanyalah gerakan yang berasal dari satu blok saja, yaitu blok kosong pada *puzzle*. Perbedaannya, pada *puzzle Rush Hour* ini akan ditelusuri setiap pergerakan yang dapat dilakukan semua blok yang ada dalam *puzzle* sehingga penelusuran tidak hanya bergantung pada blok berwarna merah. Berikut alur kerja yang dilakukan dalam penyelesaian *puzzle Rush Hour*:

1. Mengonversikan bentuk *puzzle* ke dalam matriks dan disimpan ke dalam sebuah file berekstensi txt
2. Program membaca file tersebut dan membuat sebuah objek *Puzzle* sebagai state awal.
3. Memasukkan objek tersebut kedalam *PriorityQueue* Q dan melakukan perhitungan *cost* dari simpul tersebut.
4. Mengambil sebuah elemen dengan *cost* terkecil dari *Priority Queue* Q dan melakukan pengecekan apakah state tersebut merupakan *final state*. Ada dua kondisi yang terjadi:
   a. Objek tersebut sesuai dengan *final state*
      Jika objek tersebut adalah *final state*, semua simpul hidup yang memiliki *cost* lebih besar dari objek tersebut akan dibunuh.
   b. Objek tersebut tidak sesuai dengan *final state*
      Membangkitkan simpul-simpul anak dari objek tersebut. Simpul-simpul anak yang dibangkitkan

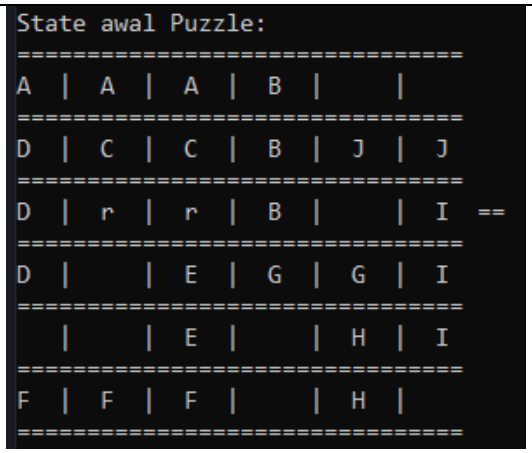
berasal dari nilai *heuristic (h(n))* semua blok yang dapat digeser pada papan puzzle milik objek puzzle saat ini ditambah dengan level kedalaman dari objek anak tersebut *(g(n))*.

5. Mengulangi langkah 4 hingga *Priority Queue* Q menjadi kosong.
6. Mencetak solusi dan langkah menuju solusi ke layar jika solusi dapat ditemukan.

VI. PENGUJIAN

Pengujian ini menggunakan puzzle dengan state awal sama seperti puzzle pada Gambar 10 dan Tabel 1.

Matriks *Puzzle*



Gambar 11. State Awal *Puzzle*

Lokasi r	(3, 2)
Lokasi final	(3, 5)
Keterangan: tanda == merupakan tanda untuk pintu keluar	

Dengan menggunakan algoritma A*, didapat solusi sebagai berikut:

Gambar 12. Solusi Langkah ke-1

Gambar 13. Solusi Langkah ke-2

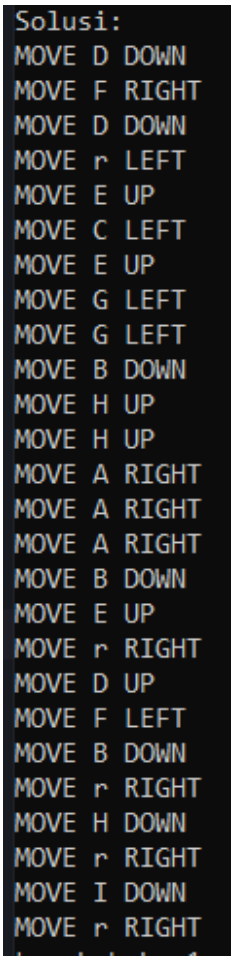
Untuk mempersingkat halaman pada makalah ini, solusi berikutnya yang akan ditampilkan merupakan tiga langkah terakhir menuju *final state*.

Gambar 14. Solusi Langkah ke-24

Gambar 15. Solusi Langkah ke-25

Gambar 15. Solusi Langkah ke-26 (Solusi)

Dan berikut merukapan langkah pertama hingga langkah terakhir tanpa menggunakan ilustrasi gambar *puzzle*.



Gambar 16. Solusi Lengkap

VII. Kesimpulan

Algoritma A* merupakan algoritma yang sangat mangkus untuk menyelesaikan berbagai persoalan. Pada makalah ini telah dibuktikan bahwa algoritma A* menggunakan fungsi *heuristic manhattan distance* yaitu jarak blok berwarna merah ke pintu keluar mampu menyelesaikan *puzzle Rush Hour* dengan cepat dan efektif.

VIII. Video Link at Youtube

Penjelasan makalah ini dapat dilihat pada pranala: https://youtu.be/9DPCX9Z2lFw

IX. Ucapan Terima Kasih

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena karunia-Nya telah memberikan kesempatan untuk menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada semua pihak yang telah membantu pengerjaan makalah ini baik secara langsung maupun tidak langsung. Secara khusus penulis mengucapkan terima kasih kepada Pak Rinaldi, selaku dosen dari Mata Kuliah Strategi Algoritma kelas 03 yang telah membantu saya untuk memahami materi-materi mengenai strategi algoritma, khususnya algoritma A*. Selain itu, penulis juga mengucapkan terima kasih kepada seluruh sumber referensi yang telah membantu penulis menyelesaikan makalah ini.

Referensi

[1] Problem Solving and Search A Star Best FS dan UCS (2018), (Online), https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/A-Star-Best-FS-dan-UCS-(2018).pdf, diakses pada tanggal 24 April 2020.

[2] Searching: Uninformed & Informed, (Online), https://socs.binus.ac.id/2013/04/23/uninformed-search-dan-informed-search/, diakses pada tanggal 24 April 2020.

[3] Munir, Rinaldi. "*Matematika Diskrit Edisi 3*". Informatika, Bandung: 2010.

[4] Heuristics,(Online),http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html, diakses pada 24 April 2020.

[5] Rush Hour Assignment, (Online), https://www.cs.princeton.edu/courses/archive/fall07/cos402/assignments/rushhour/ diakses 25 April 2020.

[6] Rush Hour Algorithm Report, (Online), https://www.cse.huji.ac.il/~ai/projects/2015/RushHour/files/report.pdf diakses 30 April 2020.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Madiun, 1 Mei 2020

Tony Eko Yuwono
13518030