# DOKUMENTASI TUGAS 1 PENGANTAR SISTEM CERDAS

# "Rush Hour" MENGGUNAKAN A*


Disusun oleh :

2017730045 - Denise Stevani Gebriella

2017730058 - Mhd.Anggie Rinovka

2017730084 - Rajasa Cikal


**UNIVERSITAS KATOLIK PARAHYANGAN**

**FAKULTAS TEKNOLOGI INFORMASI DAN SAINS**

**JURUSAN INFORMATIKA**

**BANDUNG**

**2020**

# DAFTAR ISI


**1. PENDAHULUAN** **3**

**1.1 Latar Belakang** **3**

**1.2 Rumusan Masalah** **4**

**1.3 Tujuan** **4**

**1.4 Batasan Masalah** **5**

**1.5 Metodologi** **5**

**1.6 Sistematika Pembahasan** **5**

**2 LANDASAN TEORI** **6**

**2.1 Algoritma A*** **6**

**2.2 Heuristic** **7**

**2.3 Aturan Permainan** **7**

**3 ANALISIS MASALAH** **12**

**3.1 Analisis Masalah** **12**

**4 IMPLEMENTASI** **17**

**4.1 Implementasi** **17**

**5 EKSPERIMEN** **24**

**5.1 Hasil Eksperimen** **24**

**6 KESIMPULAN** **27**

**6.1 Kesimpulan** **27**

**REFERENSI** **28**

# BAB 1

# PENDAHULUAN

## 1.1 Latar Belakang

Ada banyak jenis permainan yang bertujuan untuk melatih kemampuan otak manusia salah satunya adalah Rush Hour dimana permainan logik ini berupa blok puzzle yang harus digeser dimana balok tersebut mewakili mobil yang dijalankan dalam sebuah kotak berukuran 6x6. Tujuan dari permainan ini adalah agar mobil bisa keluar dari kemacetan tanpa ada yang saling bertabrakan dengan mengeluarkan mobil lain dari jalurnya[1].

Permainan ini bisa dimainkan secara langsung dalam bentuk fisik atau dalam komputer dengan bentuk sebuah program dengan menggunakan algoritma A*.

[1] *On the Symbolic Computation of the Hardest Configurations of the RUSH HOUR Game*, diakses dari https://link.springer.com/chapter/10.1007/978-3-540-75538-8_20#authorsandaffiliations , ditulis oleh Sébastien Collette, Jean-François Raskin, dan Frédéric Servais dan diakses pada 17 Desember 2020

## 1.2 Rumusan Masalah

Sesuai dengan latar belakang berikut adalah rumusan masalah yang akan dibahas :

1. Bagaimana cara menyelesaikan puzzle agar saat mobil dipindahkan tidak ada balok yang saling bertabrakan dan mobil yang berwarna merah bisa keluar dari puzzle
2. Bagaimana cara agar permainan ini dapat diselesaikan dengan komputer?

## 1.3 Tujuan

Berdasarkan rumusan masalah, tujuan dari tugas ini adalah sebagai berikut :

1. Untuk mencari tahu bagaimana cara agar saat memindahkan mobil tidak ada tabrakan sama sekali dan mobil berwarna merah dapat keluar dengan sukses
2. Membuat program algoritma untuk menjalankan permainan ini

## 1.4 Batasan Masalah

Batasan masalah dalam tugas ini adalah sebagai berikut :

1. Program yang dapat menyelesaikan masalah dari *Rush Hours* dengan *input* yang dimasukkan ketika dijalankan apakah berhasil atau tidak
2. Program akan mengeluarkan *output* berbentuk *String*

## 1.5 Metodologi

Langkah-langkah yang akan ditempuh untuk menyelesaikan tugas ini adalah :

1. Melakukan studi tentang *Rush Hour* dan *Heuristic*
2. Mempelajari bagaimana penggunaan algoritma A*
3. Membuat program untuk menyelesaikan masalah tersebut

## 1.6 Sistematika Pembahasan

1. Bab 1 Pendahuluan berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan
2. Bab 2 Landasan teori berisi dasar teori mengenai algoritma A*
3. Bab 3 Analisis Masalah berisi analisa bagaimana penyelesaian permainan ini
4. Bab 4 Implementasi berisi bagaimana perancangan program
5. Bab 5 Eksperimen berisi percobaan *input* yang digunakan agar *Rush Hour* berhasil, dan class diagram
6. Bab 6 Kesimpulan

# BAB 2

# LANDASAN TEORI

## 2.1 Algoritma A*

Algoritma A* atau A* *search algorithm* adalah algoritma pencarian atau *searching* dengan mencari atau menelusuri jalur terpendek antara *initial state* dan *final state*[2]. Algoritma ini menyimpan node yang dihasilkan dalam memori.

Cara kerja algoritma A* adalah sebagai berikut :

1. Algoritma ini akan dimulai dari node awal yang telah ditentukan
2. Lalu setelah node awal didapatkan, akan dicari jarak yang akan dilalui oleh node awal tersebut (akan menelusuri seluruh jalur) saat mencari jalur akan dihitung *cost* yang digunakan. Hingga akhirnya mendapatkan jarak yang terpendek menuju node tujuan
3. Setelah jarak terpendek didapatkan maka telah ditemukan jalur terpendek menuju node tujuan

[2] *What is the A* Algorithm?*, diakses dari https://www.educative.io/edpresso/what-is-the-a-star-algorithm , ditulis oleh Edpresso *Team* dan diakses pada 17 Desember 2020

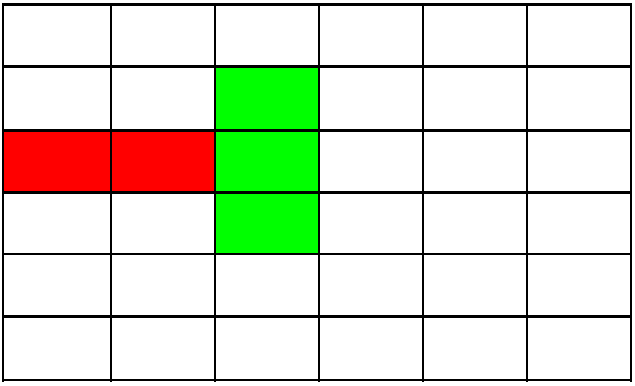
## 2.2 Heuristic

Fungsi Heuristik adalah metode untuk memberikan informasi pada algoritma *search* lain tentang arah untuk menuju suatu tujuan dimana akan ditebak *neighbor* mana yang yang akan menuju ke tujuan[3].

## 2.3 Aturan Permainan

*Rush Hour* memiliki beberapa aturan permainan diantaranya adalah

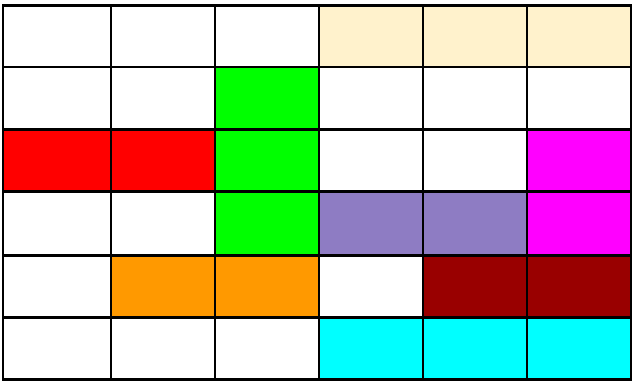
1. Satu papan terdiri dari mobil mobil, dan terdapat satu mobil yang terjebak (kedepannya akan disebut red car).



Gambar 2.1 *red car* dan mobil yang menghalangi jalannya *red car*

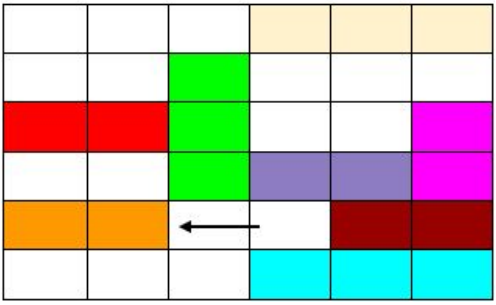
[3] *Heuristic Search*, diakses dari https://artint.info/html/ArtInt_56.html#:~:text=The%20heuristic%20function%20is%20a,readily%20obtained%20about%20a%20node. , ditulis oleh David Poole, dan Alan Mackworth dan diakses pada 17 Desember 2020

2. Di dalam papan berukuran 6x6 tersebut, terdapat mobil mobil yang diposisikan dengan posisi horizontal, dan vertikal.

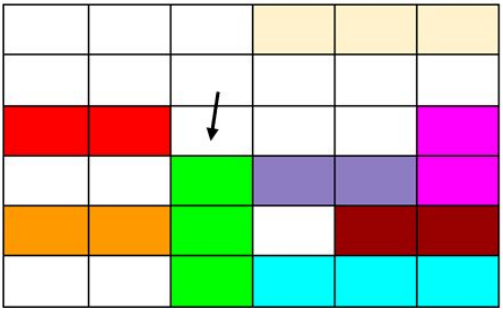


Gambar 2.2 Posisi mobil-mobil dan orientasinya

3. Mobil mobil tersebut hanya bisa digerakan ke atas-bawah, bawah-atas, kanan-kiri atau kiri-kanan, sesuai dengan orientasinya.

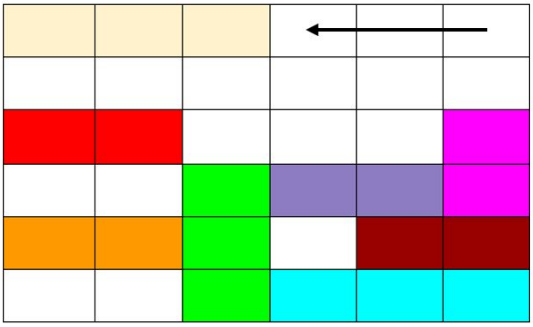


Gambar 2.3 Mobil *orange* bergerak ke arah kiri

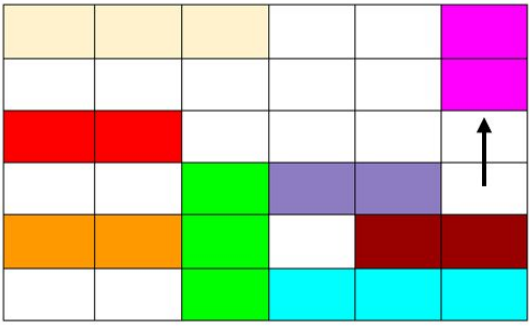


Gambar 2.4 Mobil hijau bergerak ke arah bawah

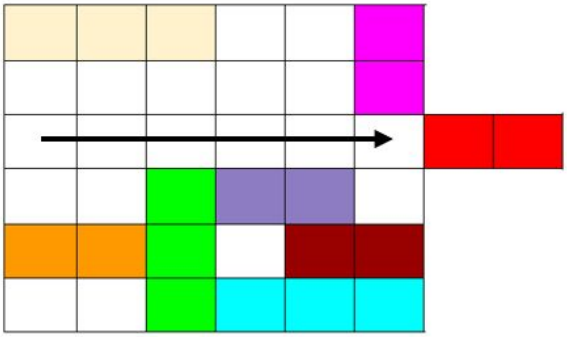
4. Tujuan dari permainan ini adalah mengosongkan jalan keluar untuk red car, agar dia bisa keluar dari titik asalnya, menuju jalan keluar yang telah ditetapkan.



Gambar 2.5 mobil berwarna pink bergerak ke arah kiri



Gambar 2.6 mobil berwarna ungu bergerak ke arah atas



Gambar 2.7 *red car* keluar dari tempat parkir(*puzzle*)

# BAB 3

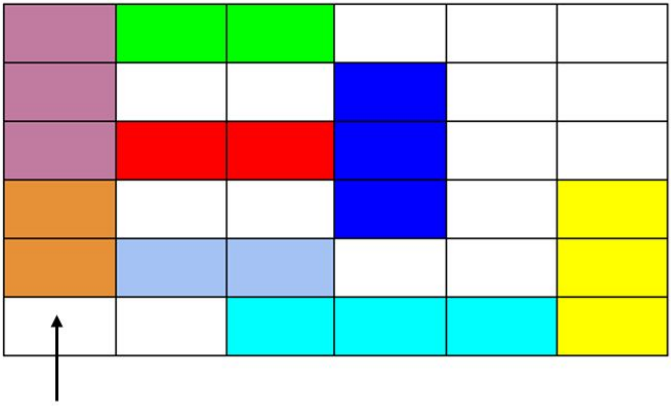
# ANALISIS MASALAH

## 3.1 Analisis Masalah

Pada bab ini akan dijelaskan analisis masalah bagaimana menyelesaikan permainan *Rush Hour* berikut adalah langkah-langkahnya :

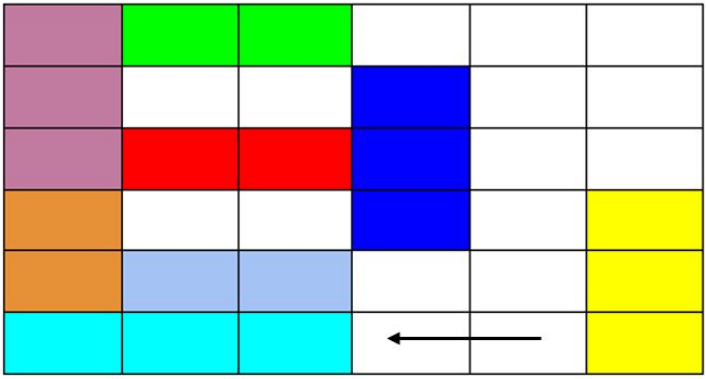
- Untuk *red car* dapat berjalan harus diperiksa apakah ada mobil lain yang menghalangi jalurnya.
- Jika ada, lihat posisi dari mobil yang menghalangi, lalu lihat apakah mobil tersebut bisa bergerak dari kiri-kanan, kanan-kiri, atas-bawah, atau bawah-atas (sesuai dengan posisi mobil tersebut apakah vertikal atau horizontal) dan jika ada mobil lain yang menghalangi jalannya mobil tersebut maka akan dicek lagi apakah mobil tersebut bisa memindahkan posisinya atau tidak.
- Jika jalur *red car* sudah kosong maka *red car* bisa maju dan keluar dari tempat parkir.

Berikut adalah visualisasi dari permainan *Rush Hour* :



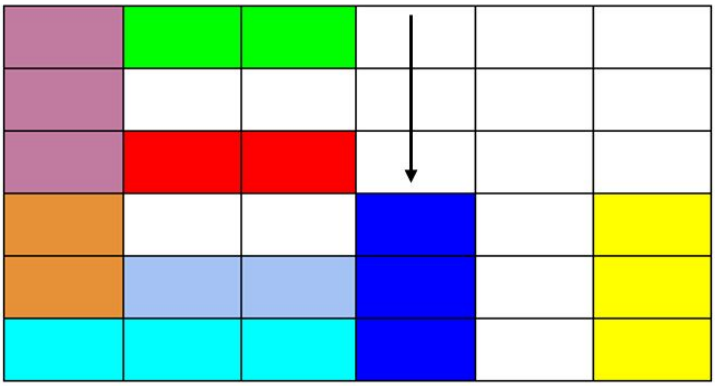
Gambar 3.1 Langkah awal permainan

1. Langkah pertama sesuai dari gambar 3.1, cek terlebih dahulu apakah didepan redcar ada *car* lain atau tidak, jika iya cek lagi apakah *car* lain berorientasi vertikal atau horizontal.



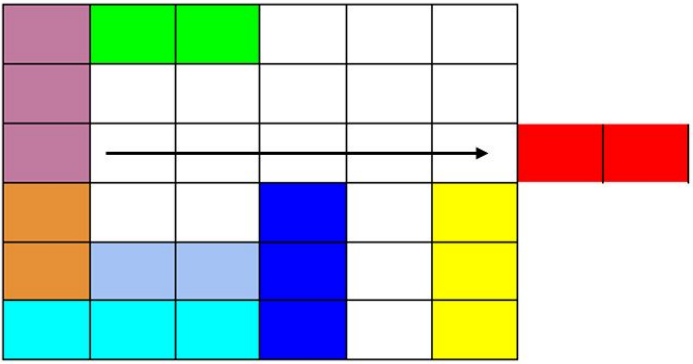
Gambar 3.2 Langkah kedua dari permainan

2. Langkah selanjutnya saat sudah mengecek apakah ada mobil yang menghalangi jalannya, maka akan mencari jalan bagaimana agar mobil merah tersebut bisa keluar dan dilihat juga apakah sebuah mobil bisa berpindah jika *lane* mereka kosong dari gambar 3.2 mobil berwarna cyan dengan posisi horizontal dipindahkan ke arah kiri dikarenakan pada awalnya posisi tersebut kosong.



Gambar 3.3 Langkah ketiga dari permainan

3. Kemudian dilihat lagi apakah masih ada mobil yang bisa berpindah, sesuai dari gambar 3.3 bisa dilihat bahwa mobil biru didepan red car bisa berpindah, maka sesuai dari ukuran mobilnya dan jalurnya, mobil biru dapat berpindah ke bawah.



Gambar 3.4 Langkah keempat dari permainan

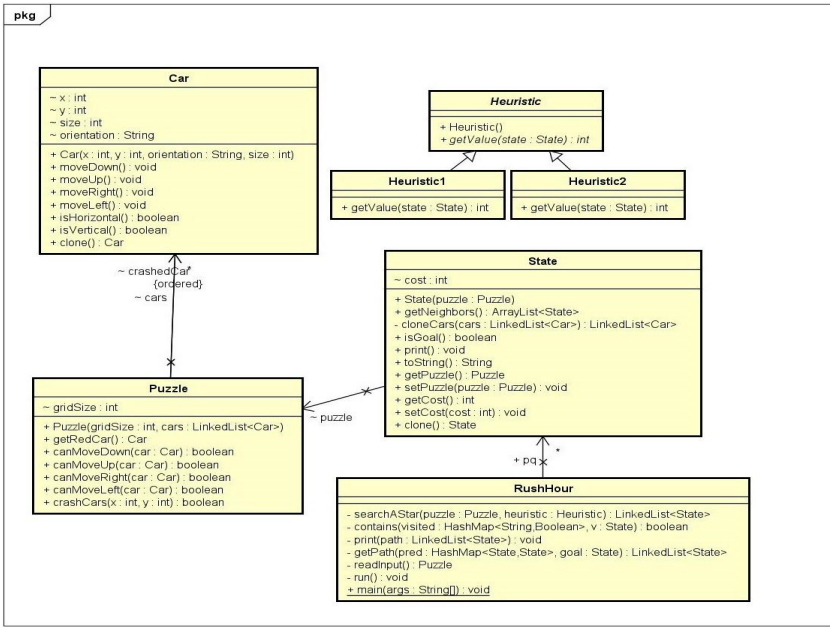
4. Terakhir bisa dilihat dari gambar 3.4 jalur untuk *red car* dapat keluar sudah kosong, dan *red car* pun bisa maju untuk keluar dari tempat parkir.

# BAB 4

# IMPLEMENTASI

## 4.1 Implementasi

Secara garis besar, program ini menjalankan 5 kelas utama, yaitu RushHour, State, Car, Heuristic, dan Puzzle, dan *inheritance* dari kelas java Heurstic yaitu Heuristic1, dan Heuristic2.




Gambar 4.1 kelas diagram dari perangkat lunak yang sudah dibuat

**a. Kelas RushHour**

Merupakan *main class* dari program Rush Hour ini.

Kelas ini memiliki 2 buah atribut yaitu :

1. in bertipe *Scanner* dimana atribut ini bertujuan untuk mengeluarkan hasil *output*.
2. pq bertipe PriorityQueue<*State*> dimana atribut ini bertujuan untuk memprioritaskan yang utama yang bertipe kelas State;

Kelas ini memiliki 7 buah metode yaitu :

1. searchAStar(*Puzzle Heuristic*) bertipe LinkedList<*State*>. Metode ini mencari cara agar *red car* bisa mencapai tujuannya dengan mencari tahu berapa ukuran *car* serta menandakan lokasi yang telah dikunjungi maupun yang belum dikunjungi.
2. contains(HashMap<*String*, *boolean*>, *State*) bertipe Boolean. Metode ini bertujuan untuk memeriksa apakah kelas State berisi value atau tidak.
3. print(LinkedList<*State*>)
   Bertipe void. Metode ini bertujuan untuk print hasil *linked list* dari kelas *State*.
4. getPath(HashMap<*State*, *State*> pred, *State*)
   Bertipe LinkedList<State>. Metode ini bertujuan untuk mendapatkan langkah langkah yang tepat
5. readInput() bertipe Puzzle, yang bertujuan untuk membaca input yang telah diisi sesuai dengan peraturan yang ada.

6. run() bertipe void, yang bertujuan menjalankan program yang telah lancar.
7. main(String[] *args*) bertipe void, yang bertujuan untuk menjalankan program dengan memanggil fungsi readInput().

**b. Kelas State**

Merupakan kelas *state* yang dimana menyatakan keadaan *car-car* yang ada.

Kelas ini memiliki 2 buah atribut yaitu :

1. puzzle bertipe *Puzzle* dimana atribut ini berfungsi untuk membuat objek *puzzle.*
2. cost bertipe *Integer* dimana atribut ini berfungsi untuk menyimpan nilai berapa banyak *cost*(langkah) yang dihabiskan dari *state* awal ke *state* tujuan.

Kelas ini memiliki 10 buah atribut yaitu:

1. getNeighbors() bertipe *ArrayList*<*State*>. Metode ini berfungsi untuk cek apakah posisi *red car* bertabrakan dengan *cloneCar* lain atau tidak, serta cek apakah *cloneCar* berada diposisi vertical atau horizontal dan cek apakah cloneCar tersebut bisa bergerak atau tidak.
2. cloneCars(*LinkedList<Car>*) bertipe *LinkedList<Car>*. Metode ini berfungsi untuk membuat sebuah *Linkedlist* yang bertipe Car.
3. isGoal() bertipe *boolean*. Metode ini berfungsi untuk cek apakah *redcar* telah mencapai tujuannya atau tidak.
4. print() bertipe *void*. Metode ini berfungsi untuk Mengeluarkan output yang telah diproses di kelas toString.

5. toString() bertipe *void*. Metode ini berfungsi untuk menerima *input* yang dimasukkan dan mengeluarkan hasilnya akan keluar dalam format *String.*
6. getPuzzle() bertipe *Puzzle*. Metode ini berfungsi untuk mengambil objek *puzzle.*
7. setPuzzle() bertipe *void*. Metode ini berfungsi untuk membuat objek *puzzle* baru.
8. getCost() bertipe *integer*. Metode ini berfungsi untuk untuk mengambil nilai *cost.*
9. setCost() bertipe *void*. Metode ini berfungsi untuk membuat nilai *cost* baru.
10. clone() bertipe *State*. Metode ini berfungsi untuk membuat *state* baru dari *puzzle*.

**c. Kelas Car**

Merupakan kelas car yang dimana menyatakan ukuran mobil serta posisi-posisinya.

Kelas ini memiliki 4 buah atribut yaitu :

1. x bertipe *integer* dimana atribut ini merupakan titik baris pada *board.*
2. y bertipe *integer* dimana atribut ini merupakan titik kolom pada *board.*
3. size bertipe *integer* dimana atribut ini merupakan ukuran dari *puzzle*.
4. orientation bertipe *String* dimana atribut ini menyatakan orientasi (posisi) dari mobil apakah vertikal atau horizontal.

Kelas ini memiliki 7 buah metode yaitu :

1. moveDown() bertipe void. Metode ini berfungsi untuk posisi atribut x berpindah ke bawah tetapi posisi atribut y tetap.
2. moveUp() bertipe *void*. Metode ini berfungsi untuk posisi atribut x berpindah ke atas tetapi posisi atribut y tetap.
3. moveRight() bertipe *void*. Metode ini berfungsi untuk posisi atribut y berpindah ke kanan tetapi posisi atribut x tetap.
4. moveLeft() bertipe *void*. Metode ini berfungsi untuk posisi atribut y berpindah ke kiri tetapi posisi atribut x tetap.
5. isHorizontal() bertipe *boolean.* Metode ini berfungsi untuk menentukan orientasi dari mobil adalah horizontal.
6. isVertical() bertipe *boolean.* Metode ini berfungsi untuk menentukan orientasi dari mobil adalah vertikal.
7. clone() bertipe *Car*. Metode ini berfungsi untuk membuat objek mobil dan membuat posisi untuk objek mobil tersebut.

**d. Kelas Heuristic**

Merupakan kelas abstrak yang berperan sebagai parent dari kelas Heuristic1 dan Heuristic2 dimana dalam kelas ini terdapat metode getValue(*State*) yang akan digunakan dikelas Heuristic1 dan Heuristic2 untuk mengambil *value*.

**e. Kelas Puzzle**

Merupakan kelas untuk menyediakan tempat untuk meletakkan mobil serta metode untuk mobil dapat berpindah posisi, dan membuat mobil yang bisa menghalangi jalan keluarnya *red car*.

Kelas ini memiliki 3 buah atribut yaitu :

1. gridSize bertipe *integer* dimana atribut ini berfungsi untuk menyatakan ukuran puzzle.
2. car bertipe *ArrayList<Car>* dimana atribut ini menyatakan object car yang sudah ada.
3. crashedCar bertipe *Car* dimana atribut ini merupakan atribut untuk membuat mobil baru untuk menghalangi jalannya *red car*.

Kelas ini memiliki 6 buah metode yaitu :

1. getCar() bertipe *Car*. Metode ini berfungsi untuk mendapatkan mobil *red car.*
2. canMoveDown(*Car*) bertipe *boolean*. Metode ini berfungsi untuk memindahkan posisi mobil kearah bawah apabila ukuran *puzzle* cukup untuk mobil dapat dipindahkan.
3. canMoveUp(*Car*) bertipe *boolean*. Metode ini berfungsi untuk memindahkan posisi mobil kearah atas apabila ukuran *puzzle* cukup untuk mobil dapat dipindahkan.
4. canMoveRight(*Car*) bertipe *boolean*. Metode ini berfungsi untuk memindahkan posisi mobil kearah kanan apabila ukuran *puzzle* cukup untuk mobil dapat dipindahkan.

5. canMoveLeft(*Car*) bertipe *boolean*. Metode ini berfungsi untuk memindahkan posisi mobil ke arah kiri apabila ukuran *puzzle* cukup untuk mobil dapat dipindahkan.
6. crashCar(*integer*, *integer*) bertipe *boolean*. Metode ini berfungsi untuk menentukan apakah ada mobil yang saling bertabrakan dan menentukan penempatan mobil yang akan menghalangi jalannya *red car* apakah akan berposisi vertikal atau horizontal dan menentukan ukuran mobil yang akan menjadi penghalang.

**f. Kelas Heuristic1**

Merupakan kelas untuk menghitung dalam sekali bermain ada berapa banyak *movement* yang dilakukan.
Kelas ini memiliki 1 buah metode yaitu :

1. getValue(*State*) bertipe *Integer*. Metode ini berfungsi untuk melihat ada berapa banyak pergerakan yang dilakukan saat permainan dijalankan dan menentukan langkah dari mobil penghalang apakah bisa berpindah atau tidak dengan memeriksa keseluruhan posisi dari mobil dalam *puzzle*.

**g. Kelas Heuristic2**

Merupakan kelas untuk mengembalikan *state* posisi awal.
Kelas ini memiliki 1 buah metode yaitu :

1. getValue(*State*) bertipe *Integer*. Merupakan metode yang mengembalikan 0 untuk mengembalikan *state* keposisi awal.

# BAB 5

# EKSPERIMEN

## 5.1 Hasil Eksperimen

Kami melakukan pengujian untuk test case ini sebanyak 6 kali, dengan hasil percobaan sebagai berikut :

1. **Percobaan Pertama**

   **.###.@**
   ****...@**
   **==..+@**
   **.**.+.**
   **...+****
   ****.+..**

   Percobaan ini berhasil dengan keluaran waktu jika menggunakan *heuristic* = 75 detik dan keluaran waktu jika tidak menggunakan *heuristic* = 373 detik.

2. **Percobaan Kedua**

   **###..+**
   **..@..+**
   **==@...**
   **+.@.****
   **+...+.**
   **###.+.**

Percobaan ini berhasil dengan keluaran waktu jika menggunakan *heuristic* = 22 detik dan keluaran waktu jika tidak menggunakan *heuristic* = 47 detik.

3. **Percobaan Ketiga**

**..@.****

****@+..**

**==@+****

**********..**

**+###****

**+..**.**

Percobaan ini gagal karena tidak terdapat pergerakan yang bisa dijalankan oleh *red car* maupun mobil yang menghalangi.

4. **Percobaan Keempat**

**==..****

**+@....**

**+@....**

**+@....**

**.....**

******....**

Percobaan ini gagal karena tidak terdapat pergerakan yang bisa dijalankan oleh *red car* maupun mobil yang menghalangi.

**5. Percobaan Kelima**

**==..++**

**+@..++**

**+@....**

**.@....**

**.....**

****....**

Percobaan ini berhasil dengan keluaran waktu jika menggunakan *heuristic* = 9 detik dan keluaran waktu jika tidak menggunakan *heuristic* = 23 detik.

**6. Percobaan Keenam**

**==..++**

**+@@.++**

**+@@+..**

**.@@++.**

****..+.**

****....**

Percobaan ini berhasil dengan keluaran waktu jika menggunakan *heuristic* = 10 detik dan keluaran waktu jika tidak menggunakan *heuristic* = 86 detik

# BAB 6

# KESIMPULAN

## 6.1 Kesimpulan

Berdasarkan percobaan kami dalam menggunakan A* untuk menyelesaikan masalah *puzzle Rush Hour* kesimpulan yang kami dapatkan adalah sebagai berikut :

1. Pada game *rush hour* metode A* cocok karena pada game rush hour kita mencari kemungkinan sebanyak mungkin agar bisa mencapai ke tujuan dan bisa mendapatkan *cost* terkecil.
2. Untuk mendapatkan hasil yang optimal dari percobaan yang ada, maka penempatan mobil-mobil (balok *puzzle*) tersebut tidak boleh berantakan serta peletakkan posisinya tidak boleh sulit.
3. Semakin sulit penempatan mobil-mobil (balok *puzzle*) maka waktu yang dibutuhkan juga akan semakin lama.

## REFERENSI

Kami menggunakan Rush Hour with A* https://cutt.ly/lhVzvlk sebagai referensi dari kode program kami.