LAPORAN PROJECT AKHIR

**KECERDASAN BUATAN**



**DISUSUN OLEH :**

**MUHAMMAD MISHBAHUL MUNIR 145150200111181**

**M. KEVIN PAHLEVI 145150200111191**

**WINNY ARDHIAN SEPTIKO 145150200111194**

**MAHARDIKA PUTRA UTAMA 145150201111166**

**Dosen Pengajar : M. Ali Fauzi, S.Kom., M.Kom.**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS BRAWIJAYA**

**MALANG**

**2016**

**DAFTAR ISI**

[**BAB I** 1](#_Toc471868308)

[**PENDAHULUAN** 1](#_Toc471868309)

[**1.1** **LATAR BELAKANG**  1](#_Toc471868310)

[**1.2** **RUMUSAN MASALAH**  1](#_Toc471868311)

[**1.3** **MANFAAT**  2](#_Toc471868312)

[**BAB II** 3](#_Toc471868313)

[**DESKRIPSI DAN PERANCANGAN SISTEM** 3](#_Toc471868314)

[**2.1** **DESKRIPSI UMUM**  3](#_Toc471868315)

[**2.2** **PEAS**  4](#_Toc471868316)

[**2.3** **FITUR – FITUR**  5](#_Toc471868317)

[**2.4** **ENVIRONMENT PROGRAM**  5](#_Toc471868318)

[**2.5** **ALGORITMA YANG DIGUNAKAN**  6](#_Toc471868319)

[**2.6** **REFERENSI**  7](#_Toc471868320)

[**BAB III** 8](#_Toc471868321)

[**IMPLEMENTASI SISTEM** 8](#_Toc471868322)

[**3.1** **IMPLEMENTASI SISTEM**  8](#_Toc471868323)

[**3.1.1 SOURCE CODE**  8](#_Toc471868324)

[**3.1.2 PENJELASAN SOURCE CODE**  11](#_Toc471868325)

[**3.1.3 PENJELASAN CLASS DAN FUNGSI / METHOD**  15](#_Toc471868326)

[**3.2** **SCREENSHOT HASIL APLIKASI**  17](#_Toc471868327)

[**BAB IV**  20](#_Toc471868328)

[**PENGUJIAN SISTEM**  20](#_Toc471868329)

[**4.1** **SKENARIO PENGUJIAN, TERUTAMA PENGUJIAN PERFORMA ALGORITMA**  20](#_Toc471868330)

[**4.2** **HASIL PENGUJIAN DAN ANALISIS**  20](#_Toc471868331)

[**4.3** **KESIMPULAN**  22](#_Toc471868332)

[**DAFTAR PUSTAKA** 23](#_Toc471868333)

[**LAMPIRAN PEMBAGIAN TUGAS** 24](#_Toc471868334)

# **BAB I**

# **PENDAHULUAN**

## **LATAR BELAKANG**

Permainan labirin atau *maze* merupakan sebuah permainan puzzle yang menggunakan labirin sebagai area permainannya. Aturan dalam permainan ini AI harus dapat mencari jalan dari posisi awal ia berada menuju posisi tujuan dan didalam labirin terdapat beberapa tembok penghalang yang menghalangi jalan AI menuju posisi tujuan. AI harus dapat mencari jalur menuju posisi tujuan diantara tembok penghalang dan AI tidak boleh keluar dari area labirin. Dalam setiap langkah, AI hanya dapat memilih satu arah dari empat arah yang tersedia yaitu utara, selatan, barat, timur. Setiap kotak dalam labirin ditandai dengan nilai x dan y, nilai ini membantu AI untuk menentukan posisinya.

Dalam program ini di-implementasikan algoritma backtracking. Pada konsep dasarnya algoritma ini bekerja mencari solusi berdasarkan ruang solusi yang ada secara sistematis namun tidak semua ruang solusi akan diperiksa, hanya pencarian yang mengarah kepada solusi yang akan diproses. Algoritma Runut Balik berbasis pada DFS (*Depth First* *Search*) sehingga aturan pencariannya akan mengikut kepada aturan pencarian DFS yaitu dengan mencari solusi dari akar ke daun (dalam pohon ruang solusi) dengan pencarian kedalam. Simpul-simpul yang sudah dilahirkan (diperiksa) dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E atau *Expand Node*.

Melalui project akhir ini diharapkan, dengan menerapkan algoritma backtracking AI bisa mengambil langkah yang tepat agar dapat sampai posisi yang dituju.

## **RUMUSAN MASALAH**

Mengimplementasi sebuah algoritma pencari jalur kedalam sebuah robot yang dapat mencari jalur dari posisi awal menuju posisi tujuan. Labirin yang dilalui oleh robot akan terdapat halangan yang menghalangi robot untuk menuju posisi tujuan, dalam sekali langkah robot hanya dapat berpindah satulangkah di antara empat arah yaitu :

* Utara : (x,y) -> (x, y-1)
* Timur : (x,y) -> (x+1, y)
* Selatan : (x,y) -> (x, y+1)
* Barat : (x,y) -> (x-1, y)

AI secara default akan memilih langkah dari utara - timur - selatan - barat

Robot hanya dapat berjalan ke tempat yang tidak memiliki halangan dan harus tetap berada didalam area labirin. Robot harus dapat mencari jalur dari posisi awal ke posisi tujuan (*solution path* atau *goal path*), jika robot menemui halangan dalam sebuah jalur robot dapat kembali dan dan mencari jalur lain. Sebagai tambahan, robot dapat menandai jalur yang ia lewati didalam labirin.

## **manfaat**

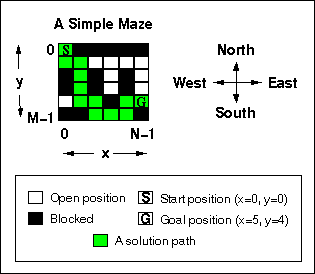
1. Dapat menyelesaikan permainan *solving a maze* dengan mencari jalur dari posisi awal ke posisi tujuan dengan mengimplementasikannya ke sebuah program.
2. Dapat memahami penyelesaian permainan *solving a maze* dengan menggunakan algoritma backtracking.

# **BAB II**

# **DESKRIPSI DAN PERANCANGAN SISTEM**

## **DESKRIPSI UMUM**

AI nantinya dapat menyelesaikan permainan solving a maze dengan menerapkan algoritma backtracking. Labirin bisa memiliki ukuran yang bermacam-macam sesuai dengan yang di inputkan oleh pengguna dan setiap kotaknya ditandai dengan nilai x dan y, dan terdapat posisi awal AI (*start state*) dan posisi tujuan (*goal state*). Labirin juga memiliki tembok penghalang yang membuat AI tidak dapat melalui suatu jalan. Berikut adalah gambar labirin tempat bermainnya :



Dalam memilih langkah AI hanya dapat memilih satu langkah diantara empat arah yang tersedia yaitu utara, timur, selatan, dan barat. AI hanya dapat berjalan di open path dan tidak dapat berjalan pada blocked atau tembok penghalang. Setiap kotak pada labirin ditandai dengan nilai x dan y. Pada labirin tersedia posisi awal dari AI dan posisi tujuan AI. Inputan dari program ini adalah sebagai berikut :

1. Nilai n yaitu jumlah baris pada labirin.
2. Nilai m yaitu jumlah kolom pada labirin.
3. Kondisi awal labirin atau tata letak open position dan blocked pada labirin, titik ( . ) menandakan *open position* dan pagar ( # ) menandakan *blocked* setiap letak path dengan yang lain dibedakan dengan spasi.
4. Posisi koordinat awal AI dengan format x y.
5. Posisi koordinat tujuan AI dengan format x y.

Kemudian hasil output dari program adalah denah labirin dengan posisi awal yang dilambangkan dengan huruf S dan posisi tujuan AI yang dilambangkan dengan huruf G, kemudian *result path* atau jalur yang di tempuh oleh AI. Jalur dengan tanda “x” adalah jalur yang sudah pernah dilalui oleh robot AI tetapi jalur ini bukan merupakan jalur menuju posisi tujuan atau bukan jalur terbaik menuju posisi tujuan, sedangkan tanda “+” adalah jalur yang menuju posisi tujuan atau jalur terbaik untuk menuju posisi tujuan.

## **PEAS**

Dalam program *solving a maze* yang diimplementasikan mempunyai PEAS (Performance Measure, Environment, Actuators, and Sensor) adalah sebagai berikut :

1. Performance Measure

*Performance Measure* yang ada pada *solving a maze* ini antara lain kecepatan robot, ketepatan menemukan solusi atau efektivitas, ketepatan dalam memilih jalur terbaik atau efisiensi, dan ketepatan dalam menentukan langkah.

1. Environment

*Environment* atau lingkungan dari robot AI adalah semua benda-benda yang ada di-labirin yaitu *open path* atau jalur yang dapat dilewati robot AI, *blocked* atau tembok penghalang atau dinding dalam, area batas labirin atau dinding luar, posisi awal robot AI, dan posisi tujuan robot AI.

1. Actuators

*Actuators* yang ada antara lain kaki robot (digunakan untuk bergerak menuju kotak selanjutnya), tangan robot (digunakan untuk menandai lintasan yang sudah dilalui).

1. Sensor

*Sensor* yang digunakan dalam robot adalah camera robot (digunakan untuk melihat kondisi sekitar robot, membedakan mana lintasan yang sudah dilewati atau belum, menentukan dinding atau bukan, menentukan mana yang merupakan tempat tujuan atau bukan).

## **FITUR - FITUR**

Fitur-fitur yang terdapat pada program *solving a maze* adalah sebagai berikut :

1. Robot AI dapat menemukan jalur dari posisi awal ke posisi akhir.
2. Robot AI dapat memilih salah satu arah dari empat arah yang disediakan.
3. Robot AI dapat mengeluarkan output jalur yang telah ditandai dengan “+” untuk lintasan solusi dan x untuk lintasan yang sudah dilewati tapi bukan menuju posisi tujuan.
4. Robot AI dapat menentukan mana *open path* dan mana *blocked path*.
5. Program dapat menentukan jumlah baris dan kolom pada labirin dengan fleksibel.
6. Program dapat menentukan kondisi labirin dengan jumlah baris dan kolom yang salam seperti yang diinputkan.
7. Program dapat menentukan koordinat awal dari posisi awal dan posisi tujuan robot AI.
8. Program dapat menampilkan kondisi labirin disertai dengan posisi awal dan posisi tujuan robot AI yang sudah ditentukan.
9. Program dapat menampilkan kondisi labirin disertai dengan lintasan solusi terbaik dari posisi awal ke posisi tujuan.

## **enviroment program**

*Environment* atau lingkungan program labirin yang terdiri dari :

1. *Open path* : jalan yang dapat dilalui oleh robot AI
2. *Blocked path* : dinding halangan yang tidak dapat dilalui oleh robot AI
3. Batas Labirin : merupakan area labirin yang membatasi area bermain dan dunia luar.
4. *Start Position* : merupakan posisi awal robot AI berada saat permainan dimulai
5. *Goal Position* : merupakan posisi tujuan dari robot AI.

## **algoritma yang digunakan**

Algoritma yang digunakan pada program adalah algoritma backtracking/ algoritma Runut Balik dan Recursive Algoritma. Algoritma Runut Balik berbasis pada DFS (Depth First Search) sehingga aturan pencariannya akan mengikut kepada aturan pencarian DFS yaitu dengan mencari solusi dari akar ke daun (dalam pohon ruang solusi) dengan pencarian kedalam. Simpul-simpul yang sudah dilahirkan (diperiksa) dinamakan sipmup hidup (live node). Simpul hidup yang sedang diperluas dinamakan simpul-E atau Expand Node. Program harus rekursif , kita melihat masalah problem mempunyai kesamaan dengan subproblem, seperti penyarian jalur yang benar dan efisien, saat pemilihan jalur kurang efisien atau tidak menemui goal maka akan di gunakan kembali method yang digunakan untuk mencari jalur yang benar

+##### start(x=1 y=1)

++...# current position(x=1 y=2)

#+####

#.####

...#.G

##...#

Untuk mencari jalan dari x=1 y=2 ke goal maka kemungkinan pencarian jalurnya adalah

* STEP(x=1, y=1) *utara*
* STEP(x=2, y=2) *timur*
* STEP(x=1, y=3) *selatan*
* STEP(x=0, y=2) *barat*

Dari fungsi rekursif kita dapat melakukan backtraking

|  |  |
| --- | --- |
| ++####  #+#..#  #+#..#  #++#.#  ###...  G...## | Sebagai contoh, misalnya posisi terakhir di x=2 y=3 di dalam maze ini maka akan memanggil STEP(x=2, y=3) |

|  |  |
| --- | --- |
| ++####  #+#..#  #+#..#  #++#.#  ###...  G...## | Pertama , mencoba mencari jalur ke goal dari posisi North dari x=2 y=3 maka memanggil PATH(x=2, y=2).  Karena pada posisi North tertutup, maka pemanggilan STEP(x=2, y=2) akan melakukan pengembalian false, dan melakukan backtraking untuk STEP(x=2, y=3) sampai ke North. |

|  |  |
| --- | --- |
| ++####  #+#..#  #+#..#  #++#.#  ###...  G...## | Selanjutnya menuju East dari x=2 y=3 , memanggil STEP(x=3, y=3).  Posisi ini tidak terbuka, maka melakukan backtraking untuk memanggil STEP(X=2,Y=3) dan dilanjutkan tahap tersebut setelah sampai di East |

|  |  |
| --- | --- |
| ++####  #+#..#  #+#..#  #++#.#  ###...  G...## | Selanjutnya akan menuju South dari x=2, y=3 dan memanggil STEP(x=2, y=4).  Posisi ini tertutup , jadi dilakukan backtraking untuk memanggil STEP(x=2, y=3) dan dilanjut kan hingga sampai di South |

|  |  |
| --- | --- |
| ++####  #+#..#  #+#..#  #++#.#  ###...  G...## | Terakhir menuju ke West dari x=2 y=3 dan memanggil STEP(x=1, y=3).  Posisi ini tertutup, jadi akan melakukan backtraking unutk memanggil STEP(x=2, y=3) dan dilanjut hingga sampai di West.  Sejak West merupakan arah terakhir untuk pencarian dari x=2 y=3 , maka akan dihilangkan tandanya di x=2 y=3 dan melakukan backtracking to pemanggilan sebelumnya yaitu STEP(x=1 y=3). |

## **referensi**

https://www.cs.bu.edu/teaching/alg/maze/

# **BAB III**

# **IMPLEMENTASI SISTEM**

## **Implementasi SISTEM**

### **3.1.1 SOURCE CODE**

|  |  |
| --- | --- |
| MazeRunner.java | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190 | package projectakhir;  import java.util.Scanner;  public class MazeRunner {  public int counter = 0;  private final static int MAX\_VALUE = 1000;  int best\_solution = MAX\_VALUE;  public char[][] maze;  public char[][] output\_maze = null;  public int start\_x, start\_y;  Scanner input = new Scanner(System.in);    public void inputMaze(){  System.out.print("Input n : ");  int n = input.nextInt();  System.out.print("Input m : ");  int m = input.nextInt();    System.out.println();    this.maze = new char[n][m];    System.out.println("Input maze : ");  for (int i = 0; i < this.maze.length; i++) {  for (int j = 0; j < this.maze[i].length; j++) {  this.maze[i][j] = input.next().charAt(0);  }  }  System.out.println();    System.out.println("Input start coordinat : ");  int goal\_y = input.nextInt();  int goal\_x = input.nextInt();  this.maze[goal\_x][goal\_y] = 'S';    System.out.println("Input goal coordinat : ");  this.start\_y = input.nextInt();  this.start\_x = input.nextInt();  this.maze[this.start\_x][this.start\_y] = 'G';    System.out.println();  }    // Membuat salinan maze  private void cloneMaze() {  this.output\_maze = new char[maze.length][maze[0].length];  for (int x = 0; x < maze.length; x++) {  for (int y = 0; y < maze[x].length; y++) {  this.output\_maze[x][y] = this.maze[x][y];  }  }  }  // Mulai dari lokasi (x,y), kemudian mencari solusi  public void solve() {  this.best\_solution = MAX\_VALUE;  if (step(this.start\_x, this.start\_y, 0) != MAX\_VALUE) {  this.output\_maze[this.start\_x][this.start\_y] = 'G';  }  }  // Metode Backtraking untuk maze  public int step(int x, int y, int count) {  counter++;    //mencetak setiap langkah  //System.out.println(this.toString());    // Langkah invalid - menembus dinding luar  if (x == -1 || x == this.maze.length || y == -1 || y == this.maze.length) {  return MAX\_VALUE;  }    //Langkah valid - menemukan tempat goal  if (this.maze[x][y] == 'S') {  this.best\_solution = count;  this.cloneMaze();  return count;  }  // Langkah invalid - menembus dinding dalam  if (this.maze[x][y] == '#' || this.maze[x][y] == '+') {  return MAX\_VALUE;  }    // Langkah invalid - sudah menemukan solusi terbaik  if (count == this.best\_solution) {  return MAX\_VALUE;  }  // Metode Backtracking    // Tandai lokasi karena merupakan lintasan menuju goal  this.maze[x][y] = '+';  int result = MAX\_VALUE;  int new\_result = MAX\_VALUE;  // Coba ke Utara  new\_result = step(x - 1, y, count + 1);  if (new\_result < result) {  result = new\_result;  }    // Coba ke Timur  new\_result = step(x, y + 1, count + 1);  if (new\_result < result) {  result = new\_result;  }    // Coba ke Selatan  new\_result = step(x + 1, y, count + 1);  if (new\_result < result) {  result = new\_result;  }    // Coba ke Barat  new\_result = step(x, y - 1, count + 1);  if (new\_result < result) {  result = new\_result;  }  // Tandai lintasan yang sudah dilintasi tapi buka lintasan menuju goal  maze[x][y] = 'x';  if (result < MAX\_VALUE) {  return result;  }  //Deadend - Lokasi bukan bagian dari solusi untuk mencapai goal  // Kembali  return MAX\_VALUE;  }  @Override  public String toString() {  String output = "";  for (int x = 0; x < this.maze.length; x++) {  for (int y = 0; y < this.maze[x].length; y++) {  output += this.maze[x][y] + " ";  }  output += "\n";  }  return output;  }  public String toStringSolution() {  if (this.output\_maze == null) {  return "No Solution!";  }    String output = "";  for (int x = 0; x < output\_maze.length; x++) {  for (int y = 0; y < output\_maze[x].length; y++) {  output += this.output\_maze[x][y] + " ";  }  output += "\n";  }  return output;  }  public static void main(String[] args) {  MazeRunner maze = new MazeRunner();    maze.inputMaze();    System.out.println("Maze with start and goal position :");  System.out.println(maze);    maze.solve();    System.out.println("Result path :");  System.out.println(maze.toStringSolution());  }  } |

### **3.1.2 PENJELASAN SOURCE CODE**

|  |  |
| --- | --- |
| MazeRunner.java | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190 | Deklarasi package  Import library Scanner  Deklarasi class MazeRunner  Deklarasi dan inisialisasi variable counter  Deklarasi dan inisialisasi variable MAX\_VALUE  Deklarasi dan inisialisasi variable best\_solution  Deklarasi variable char  Deklarasi variable output\_maze  Deklarasi variable start\_x dan start\_y  Instansiasi object Scanner  Deklarasi method inputMaze  Mencetak “Input n : ”  Deklarasi dan inisialisasi variable n  Mencetak “Input m : ”  Deklarasi dan inisialisasi variable m  Mencetak baris kosong  Inisialisasi panjang isi variable maze    Mencetak “Input maze : ”  Looping variable maze  Looping variable maze  Inisialisasi variable maze      Mencetak baris kosong    Mencetak "Input start coordinat : “  Deklarasi dan inisialisasi variable goal\_y  Deklarasi dan inisialisasi variable goal\_x  Inisialisasi variable maze index goal\_x dan goal\_y dengan ‘S’    Mencetak "Input goal coordinat : "  Deklarasi dan inisialisasi variable start\_y  Deklarasi dan inisialisasi variable start\_x  Inisialisasi variable maze index start\_x dan start\_y dengan ‘G’    Mencetak baris kosong      Komentar method  Deklarasi method cloneMaze  Inisialisasi panjang variable output\_maze  Looping variable maze  Looping variable maze  Inisialisasi variable output\_maze dengan maze  Komentar method  Deklarasi method solve  Inisialisasi variable best\_solution  Pengecekan kondisi dengan membandingkan return method step  Inisialisasi variable ouput maze dengan ‘G’    Komentar method  Deklarasi method step  Inisialisasi variable counter    Komentar  Komentar untuk mematikan pernyataan    Komentar  Pengecekan jika indek sama dengan dinding akan return nilai MAX\_VALUE  Mengembalikan nilai MAX\_VALUE      Komentar  Pengecekan jika tempat maze adalah ‘S’  Inisialisasi variable best\_solution  Memanggil method cloneMaze  Mengembalikan nilai count  Komentar  Pengecekan apakah tempat maze ‘#’ atau ‘+’  Mengembalikan nilai MAX\_VALUE    Komentar  Pengecekan variable count  Mengembalikan nilai MAX\_VALUE  Komentar    Komentar  Inisialisasi variable maze  Inisialisasi variable result  Inisialisasi variable new\_result  Komentar untuk arah utara  Inisialisasi variable new\_result dengan method step  Pengecekan kondisi new\_result  Inisialisasi variable result    Komentar untuk arah timur  Inisialisasi variable new\_result dengan method step  Pengecekan kondisi new\_result  Inisialisasi variable result    Komentar untuk arah selatan  Inisialisasi variable new\_result dengan method step  Pengecekan kondisi new\_result  Inisialisasi variable result    Komentar untuk arah barat  Inisialisasi variable new\_result dengan method step  Pengecekan kondisi new\_result  Inisialisasi variable result  Komentar  Inisialisasi variable maze  Pengecekan kondisi result  Mengembalikan nilai result  Komentar  Komentar  Mengembalikan nilai MAX\_VALUE  Deklarasi method overriding toString  Deklarasi dan inisialisasi variable output  Looping variable maze  Looping variable maze  Inisialisasi variable output  Inisialisasi variable output  Mengembalikan nilai output  Deklarasi method toStringSolution  Pengecekan nilai output\_maze  Mengembalikan nilai dengan isi "No Solution!"    Deklarasi dan inisialisasi variable output  Looping variable output\_maze  Looping variable output\_maze  Inisialisasi variable output  Inisialisasi variable output  Mengembalikan nilai output  Deklarasi method main  Intansiasi object dari kelas MazeRunner    Memanggil method inputMaze    Mencetak "Maze with start and goal position :"  Mencetak kondisi maze    Memanggil method solve  Mencetak "Result path :"  Mencetak kondisi maze disertai solusi |

### **3.1.3 PENJELASAN CLASS DAN FUNGSI / METHOD**

Kelas yang terdapat pada program :

* Class MazeRunner

Class MazeRunner ini merupakan kelas tunggal yang berisikan method-method yang digunakan dalam menyelesaikan permasalahan maze. Class ini juga ada main method yang digunakan untuk menjalankan program maze.

Method-method yang terdapat pada class MazeRunner :

* Method inputMaze

Method inputMaze ini digunakan untuk memasukkan kondisi maze mulai dari panjang dan lebar maze, kondisi maze, koordinat awal robot jalan, koordinat tujuan.

* Method cloneMaze

Method cloneMaze digunakan untuk melakukan penyalinan kondisi maze ke sebuah variable baru, dimana variable baru ini akan menyimpan kondisi akhir dari maze tersebut.

* Method solve

Method solve digunakan untuk memulai menjalankan penyelesaian permasalahan untuk mencari solusi akhir. Dalam method ini akan memanggil method step yang digunakan untuk mencari solusi terbaik untuk mencapai goal.

* Method step

Method step digunakan untuk menyelesaikan solusi maze untuk mencari goal dengan menggunakan method backtracking. Backtracking disini juga memanfaatkan method recursive untuk membantu proses penyelesaian. Dalam method ini terdapat kondisi untuk mengecek kapan robot harus backtracking yakni ketika menabrak dinding luar, dinding dalam, dan juga lintasan yang sudah dikunjungi. Penentuan arah gerak robot juga terdapat pada method ini dimana robot akan bergerak ke UTARA, jika tidak bisa akan ke TIMUR, jika tidak bisa akan ke SELATAN, jika tidak bisa akan ke BARAT.

* Method toString

Method toString digunakan untuk mencetak kondisi awal maze atau juga bisa digunakan untuk mencetak kondisi perlangkah robot dalam dari dalam maze itu sendiri.

* Method toStringSolution

Method toStringSolution digunakan untuk mencetak kondisi akhir maze yang sudah disertai dengan lintasan solusi yang merupakan lintasan terbaik.

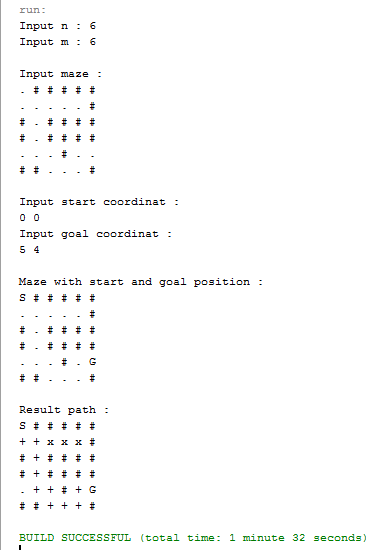
* Method main

Method main digunakan untuk menjalankan program maze yang akan menginputkan kondisi-kondisi maze kemudian akan ditampilkan solusinya.

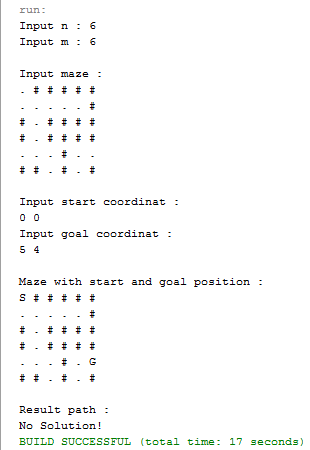
## **SCREENSHOT HASIL APLIKASI**

Berikut ini adalah screenshot dari output pada program :

* Solusi Ditemukan



* Solusi Tidak Ditemukan



# **BAB IV**

# **PENGUJIAN SISTEM**

## **SKENARIO PENGUJIAN, TERUTAMA PENGUJIAN PERFORMA ALGORITMA**

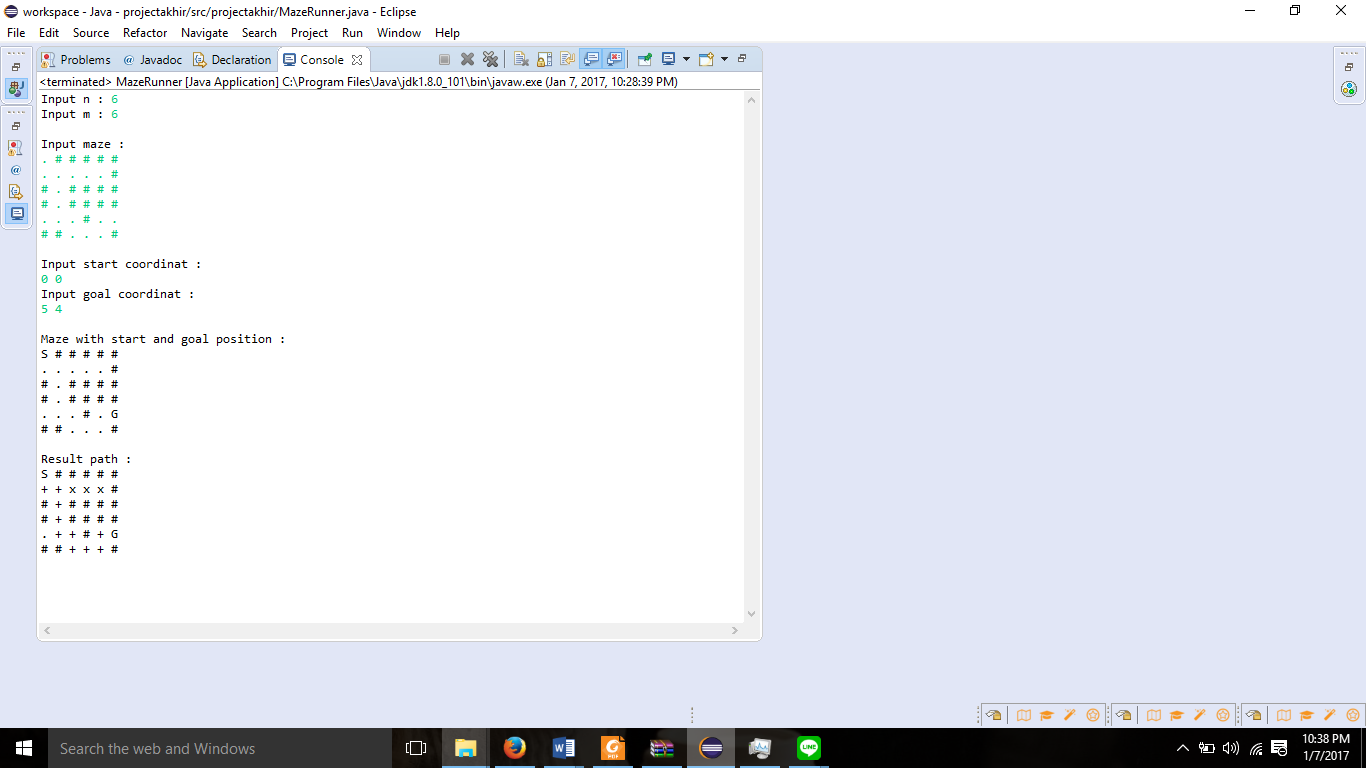
Program aplikasi di-implementasikan dengan bahasa pemrograman java dengan aplikasi netbeans. Tujuan dari pengujian ini adalah mendapatkan hasil pemilihan jalur yang optimal dan mencoba beberapa *test case* untuk menguji performa algoritma backtracking pada permainan *solving a maze*. Berikut ini beberapa *test case* yang akan diujikan :

1. Lintasan labirin yang terdapat backtracking .
2. Lintasan labirin tanpa backtracking.
3. Lintasan labirin yang memiliki dua jalur menuju tujuan.
4. Lintasan labirin yang tidak memiliki solusi.

## **hasil pengujian dan analisis**

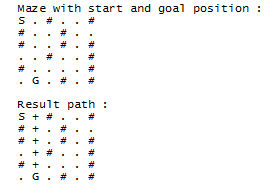
Dari beberapa *test case* yang diujikan, hasil yang didapat oleh AI adalah sebagai berikut :

1. Lintasan labirin yang terdapat backtracking



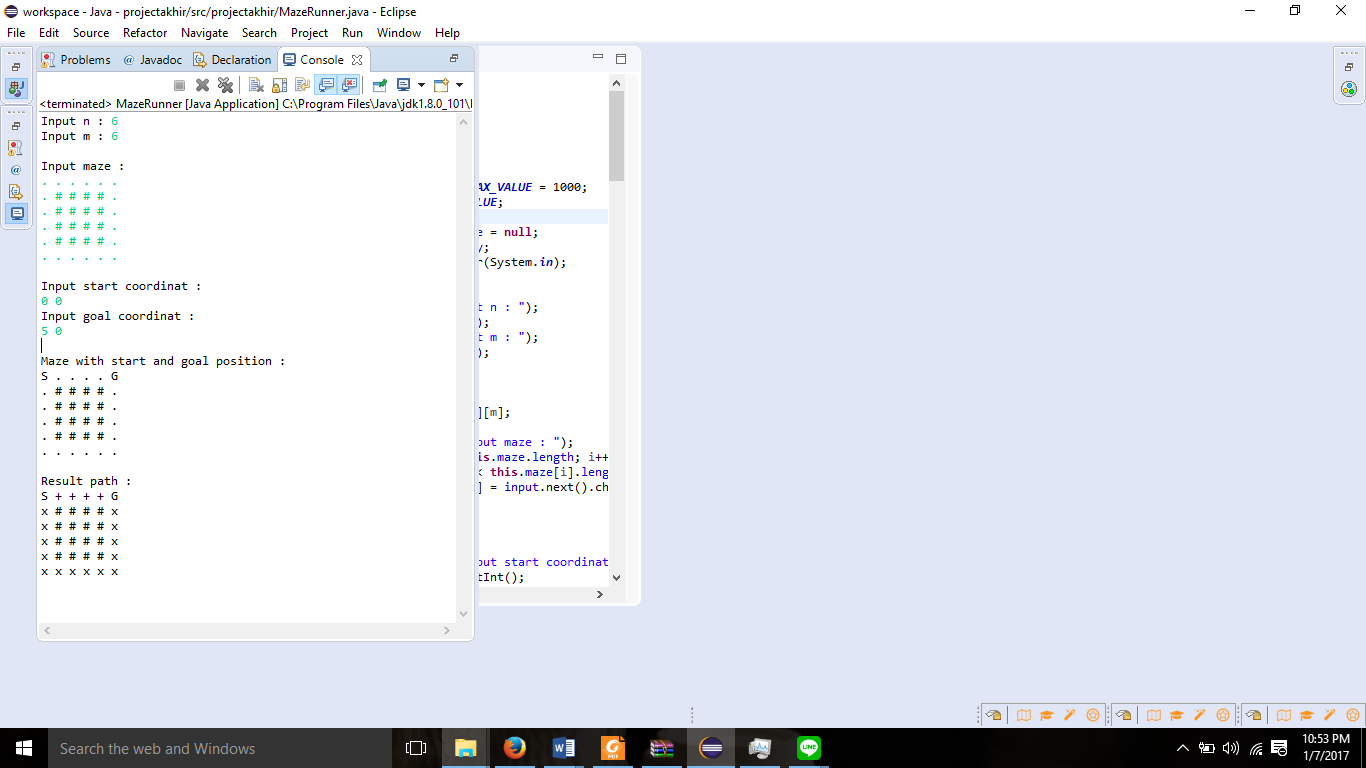
Pada denah labirin terdapat jalan buntu yang nantinya dilalui oleh robot AI yaitu pada koordinat 1, 1. AI akan memilih jalur ke timur terlebih dahulu karena robot AI memilih langkah dengan urutan utara - timur - selatan - barat, kemudian robot AI akan berjalan terus hingga menuju ke koordinat 4, 1 dimana pada koordinat tersebut tidak ada jalan. Disini robot AI akan menjalankan algoritma backtracking hingga kembali ke koordinat 1, 1 dan memilih langkah ke selatan. Robot AI dapat menemukan jalur menuju posisi tujuan.

1. Lintasan labirin tanpa backtracking



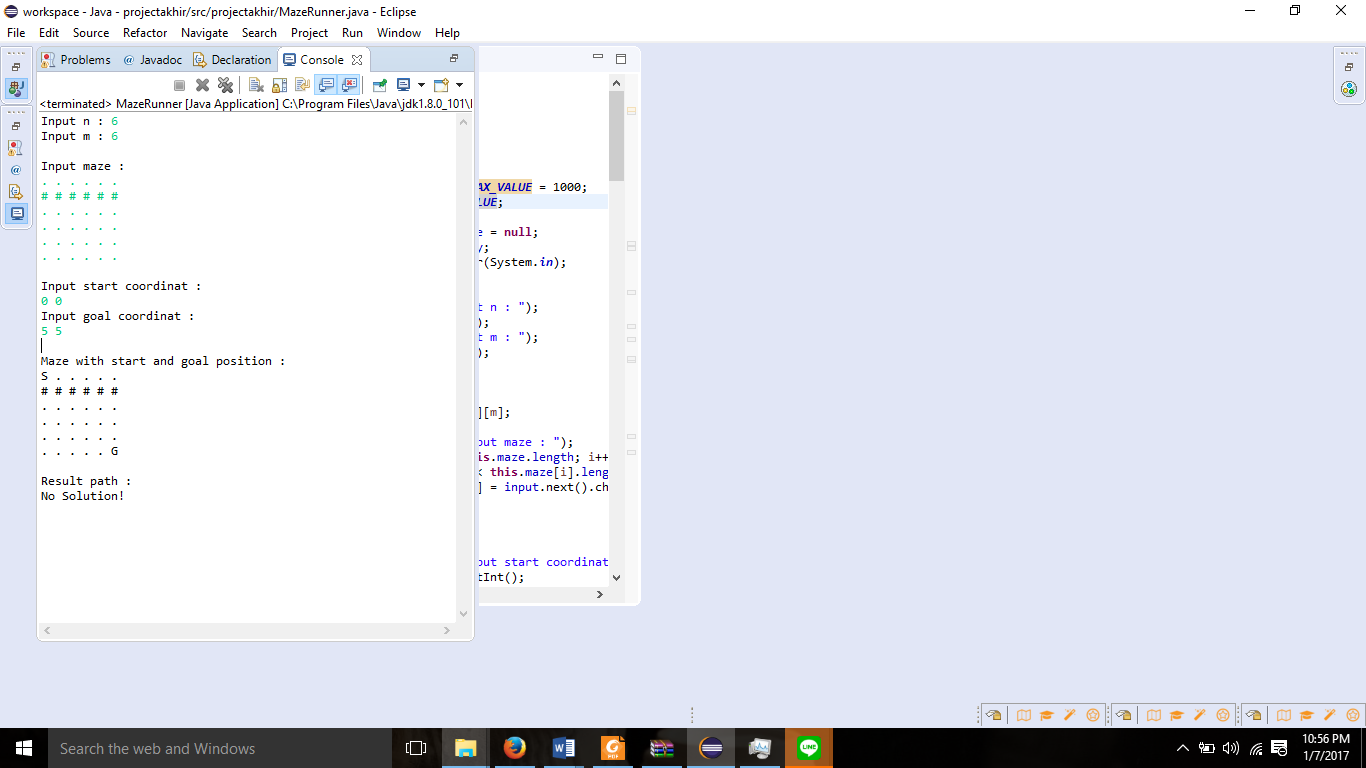
Pada *test case* dimana labirin tidak terdapat backtracking, robot AI dapat menemukan jalur menuju ke posisi tujuan.

1. Lintasan labirin yang memiliki dua jalur menuju tujuan



Pada *test case* dimana pada labirin terdapat 2 buah jalur menuju posisi tujuan, robot AI dapat menyelesaikan puzzle dan memilih jalur terpendek untuk menuju posisi tujuan.

1. Lintasan labirin yang tidak memilikik solusi.



Pada *test case* dimana labirin tidak terdapat jalur menuju posisi tujuan, robot AI mengeluarkan outputan yang menyatakan tidak ada solusi pada permainan *solving a maze*.

## **kesimpulan**

Dari hasil uji empat *test case* yang dibuat dapat diambil kesimpulan bahawa AI dapat menyelesaikan suatu permainan solving a maze yang labirinnya terdapat jalan buntu atau jalan yang membutuhkan algoritma backtracking, AI dapat menyelesaikan permainan dan menampilkan result path dengan benar. Pada permainan *solving a maze* dimana labirinnya tidak mempunyai jalan buntu dan tidak membutuhkan algoritma backtracking, AI dapat menyelesaikan permainan dan menampilkan *result path* dengan benar. Pada permainan solving a maze yang labirinnya memiliki dua buah jalur menuju posisi tujuan, AI dapat menyelesaikan permainan dan memilih jalur yang terpedek dari dua jalur yang tersedia serta AI dapat menampilkan *result path* dengan benar. Pada permainan *solving a maze* yang labirinnya tidak memiliki jalur menuju posisi tujuan, AI menampilkan peringatan bahwa solusi tidak ditemukan. Jadi, dapat disimpulkan bahwa program implementasi dari *solving a maze* menyelesaikan berbagai permasalahan yang ada dan juga dapat memilih solusi terbaik, dengan ini dapat dikatakan bahwa program yang diimplementasikan sudah efektif dan efisien diliat dari pengujian sebelumnya.

# **DAFTAR PUSTAKA**

Vincent Theophilus Ciputra. 2015. *Makalah IF2211 Strategi Algoritma.*

Robert I. Pitts. 1993-2000. *Recursion: Solving a Maze.* https://www.cs.bu.edu/teaching/alg/maze/

# **LAMPIRAN PEMBAGIAN TUGAS**

1. MUHAMMAD MISHBAHUL MUNIR

* Koding Program
* Laporan (Bab 3 : Implementasi system, penjelasan class dan fungsi, screenshot program)

1. M KEVIN PAHLEVI

* Laporan (Bab 2: Deskripsi umum, Peas, fitur-fitur, environment program, algoritma yang digunakan)

1. WINNY ARDHIAN SEPTIKO

* Video
* Laporan (Bab 1: rumusan masalah, manfaat)

1. MAHARDIKA PUTRA UTAMA

* Laporan (Bab 4 : Skenario pengujian, hasil pengujian, Kesimpulan. Bab 1 : latarbelakang)