ALGORITMA *CELLULAR AUTOMATA* (CA) DAN *BACKTRACKING* UNTUK SIMULASI PENCARIAN JALAN PADA *MAZE*

Syopiansyah Jaya Putra, Yusuf Durachman, M. Qomarul Huda
Fakultas Sains & Teknologi, UIN Syarif Hidayatullah Jakarta
Fakultas Sains & Teknologi, UIN Syarif Hidayatullah Jakarta
Fakultas Sains & Teknologi, UIN Syarif Hidayatullah Jakarta
Jl. Ir. H. Juanda, Jakarta
ssy@syopian.net

Abstract

Human being wishes to finish problems and gain advantages as much as possible with efficiency of resources.The research discusses and compares the use of algorithms: backtracking and cellular automata (CA) to look for the best way and solution. The main objective of this research is to learn the characteristic of CA and backtracking methos with their implementation at seeking walke at maze. It was found that CA method is more effective than backtracking method.

Keywords: *Cellular Automata (CA), Finite Automata (FA), Backtracking*

Pendahuluan

Setiap manusia ingin menyelesaikan permasalahan yang dihadapi dengan secepat-cepatnya dan mendapatkan keuntungan sebanyak-banyaknya dengan mengefisiensikan sumber daya yang dimiliki terhadap batasan-batasan yang ditemui pada suatu masalah. Saat ini permainan-permainan yang ada juga memberikan permasalahan-permasalahan pada penggunanya dan sangat mungkin pula terjadi di kehidupan nyata (Munir, 2004).

Selanjutnya Munir (2004) menjelaskan bahwa, dunia permainan juga adalah salah satu implementasi dari bidang informatika. Perkembangan permainan pada masa kini sudah sangat jauh, namun sebuah algoritma yang selalu menjadi dasar dari semua permainan adalah algoritma *backtracking* atau algoritma runut-balik. Algoritma runut-balik sendiri adalah algoritma yang berbasis pada *Depth First Search* (DFS) untuk mencari solusi persoalan secara lebih baik.

Rumusan Masalah

Adapun rumusan masalah yang akan dibahas adalah:

1) Bagaimana metode *backtracking* dan metode CA diimplementasikan*,*
2) Bagaimana kedua metode tersebut dapat menyelesaikan masalah *maze,*
3) Bagaimana kinerja metode CA dibandingkan dengan metode *backtrack* dalam menyelesaikan masalah *maze*.

Batasan Masalah

Agar ruang lingkup (*scope*) penelitian tidak terlalu luas, maka perlu adanya pembatasan masalah yang akan dibahas, yaitu:

1) *Maze* yang digunakan adalah *mase* 2-dimensi,
2) *Maze* yang digunakan adalah *maze orthogonal,* yang berarti jalan pada *maze* hanya dapat berpotongan secara tegak lurus.

Tujuan dan Manfaat

Penelitian ini bertujuan untuk mempelajari karakteristik dari model *backtracking* dan model CA bila diterapkan pada suatu masalah.

Sedangkan manfaat penelitian ini adalah:

1) Mengetahui metode CA dapat menyelesaikan masalah secara lebih efektif dan efisien daripada metode *backtrack,*
2) Memberikan referensi bagi semua pihak untuk mengetahui lebih lanjut tentang teori-teori yang berhubungan dengan *Backtracking* dan CA,

Tinjaun Teori

Deskripsi Umum Algoritma *Cellular Automata* (CA)

CA pertama kali dikenalkan oleh John Von Neumann pada tahun 1949 (Schatten, 1999), sebagai model formal dari *self-reproducing organisms* (Sarkar, 2000). Dalam modelnya tersebut, Neumann mampu membatasi mesin abstrak yang memiliki kamampuan untuk menurunkan karakteristik dari *self-reproducing organisms.*

Sarkar (2000) mendefinisikan *cellular automata* berdasarkan arti leksikalnya, yang berasal dari dua kata, yaitu *cellular* dan *automata. Cellular* berarti sistem yang terdiri atas sel-sel. Arti sel pada *cellular automata* hampir sama dengan pengertian sel secara biologis, yaitu sebagai satuan fungsional terkecil penyusun tubuh makhluk hidup. Hanya saja, dalam hal ini sel tidak menyusun tubuh makhluk hidup, melainkan menyusun sistem suatu mesin abstrak. Sel-sel tersebut merupakan bagian terkecil dari *cellular automata* yang masih dapat berfungsi secara atomik.

CA adalah *automata* yang terdiri atas sel-sel. Tiap sel tersebut adalah juga suatu *automaton. Automaton* tersebut dapat berjenis apa saja dan dapat sama ataupun berbeda antar selnya. (Toffoli; Tomasso; Norman Margolus, 1991).

Menurut Weimer (1996) Secara formal, CA dapat didefinisikan sebagai 5-tuple. Kelima tuple tersebut adalah :

L : Geometri

Q : Himpunan berhingga *state* (U_{Vsel}{*state*})

N : Himpunan sel yang mempengaruhi *state* sel tersebut pada langkah berikutnya.

δ : Q^{n+1} Q. (Q^{n+1} : tuple yang terdiri atas *state* dirinya sendiri dan *state* sel-sel *neighbours*-nya).

C_0 : Konfigurasi awal sistem.

Bagian *Cellular Automata*

1. Geometri , CA dapat dituliskan dalam bentuk *graph.* Tiap sel adalah *vertex,* sedangkan hubungan *neighbour* digambarkan denga *edge.*
2. State Set, *State set* adalah himpunan *state* yang mungkin. Setiap set memiliki *state set. State set* ini haruslah berhingga (*finite,* terbatas) dan terhitung (*countable,* diskret).
3. Neighbourhood, Menurut Neumann (1966); Toffoli (1987) menjelaskan bahwa *Neighbourhood* dari suatu sel A adalah sel-sel sekitar yang secara langsung berpengaruh pada perubahan *state* sel A tersebut.
4. Fungsi Transisi, Tiap sel dalam suatu CA mempunyai fungsi transisi. Pada tiap *timestep,* tiap sel mengambil *state* dirinya sendiri dan *state neighbours*-nya untuk dijadikan *input* bagi fungsi transisi

Karakteristik *Cellular Automata*

1. Sistem Diskrit yang Dinamis, Sistem dinamis ada-lah sistem yang mengalami perubahan seiring dengan berjalannya waktu
2. *Locality, updeting state* suatu sel bergantung pada fungsi transisi yang dimilikinya, *state* dirinya sendiri, dan *state neighbours*-nya.
3. *Parallelism,* Setiap sel melakukan *updeting state* serentak secara bersama-sama
4. *Emergent,* Tiap–tiap sel penyusun CA hanya melakukan fungsi-fungsi sederhana yang sepertinya tidak terlalu bermanfaat.

Deskripsi Umum Algoritma *Backtracking*

Algoritma *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Dalam perkembangannya beberapa ahli seperti RJ Walker, Golomb, dan Baumert menyajikan uraian umum tentang *backtracking* dan penerapannya dalam berbagai persoalan dan aplikasi. *backtracking* merupakan perbaikan dari algoritma *brute force search* (BFS), secara sistematis mencari solusi persoalan di antara semua kemungkinan yang ada. Hanya pencarian yang mengarah ke solusi saja yang dikembangkan, sehingga waktu pencarian dapat dihemat. Runut-balik lebih alami dinyatakan dalam algoritma rekursif (Suryadi, 1995).

Prinsip Pencarian Solusi dengan Metode *Backtracking*

Langkah-langkah pencarian solusi pada pohon ruang status yang dibangun secara dinamis adalah :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian mendalam (DFS). Simpulsimpul yang sudah dilahirkan dinamakan simpul hidup (*live*

node). Simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.

2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*). Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*). Simpul yang sudah mati tidak akan pernah diperluas lagi.
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orang tua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

Tool-Tool Rancangan

Diagram Alur (*Flowchart*)

Menurut Pressman (2002), Komputer membutuhkan hal-hal yang terperinci, maka bahasa pemrograman bukan merupakan alat yang boleh dikatakan baik untuk merancang sebuah algoritma awal. Alat yang banyak dipakai untuk membuat algoritma adalah diagram alur. Diagram alur dapat menunjukan secara jelas arus pengendalian algoritma, yakni bagaimana rangkaian pelaksanaan kegiatan. Suatu diagram alur memberikan gambaran dua dimensi berupa simbol-simbol grafis.

Unified Modelling Language (UML)

UML (*Unified Modelling Language*) pertama kali diperkenalkan pada tahun 1990-an ketika Grady Booch, Ivar Jacobson dan James Rumbaugh mulai mengadopsi ide-ide serta kemampuan-kemampuan tambahan dari masing-masing metodanya dan berusaha membuat metodologi terpadu yang kemudian dinamakan UML (*Unified Modelling Language*) (Nugroho, 2005).

Unified Modelling Language (UML) menurut Hermawan (2004:7) adalah bahasa standar yang digunakan untuk menjelaskan dan memvisualisasikan artifak dari proses analisis dan desain sistem berorientasi objek.

Diagram UML

1) *Use Case Diagram*

Use Case adalah teknik untuk merekam persyaratan fungsional sebuah sistem. *Use Case* mendeskripsikan interaksi tipikal antara para pengguna sistem dengan sistem itu sendiri, dengan memberi sebuah narasi tentang bagaimana sistem tersebut digunakan (Fowler, 2005).

Use Case Diagram menggambarkan suatu kumpulan dari beberapa *use case* dan *actors* dan hubungan mereka. Diagram ini sangat penting dalam mengatur dan mencontohkan perilaku dari sebuah sistem (Booch, 1998).

2) *Sequence Diagram*

Sebuah *sequence diagram*, secara khusus, menjabarkan *behaviour* sebuah skenario tunggal. Diagram tersebut menunjukkan sejumlah objek contoh dan pesan-pesan yang melewati objek-objek ini di dalam *use case* (Fowler, 2005: 81).

3) *Class Diagram*

Class Diagram mendeskripsikan jenis-jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat diantara mereka. *Class Diagram* juga menunjukkan properti dan operasi sebuah *class* dan batasan-batasan yang terdapat dalam hubungan-hubungan objek tersebut (Fowler, 2005:53).

Class diagram menunjukan hubungan antar *class* yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.

Studi Sejenis

Yacoubi dan Jacewicz (2006), menjelaskan bahwa untuk mendisain sebuah transisi lokal *(local state transitions*) dalam CA agar bisa melaksanakan tugas-tugas global yang spesifik, pun cukup sulit untuk lepas dari spesi-

fikasi *automaton mikroskopic* biasa kepada sebuah diskripsi prilaku global *(global behaviour)* yang lebih pantas. Paper ini bertujuan untuk mendemonstrasikan kemungkinan untuk menemukan aturan transisi yang terbaik, berbarengan dengan mengkorespondensikan lingkungan sekitar *(corresponding neighbourhood)* agar saluran automata mampu menyelesaikan tugas-tugas yang sudah dibebankan dengan baik melalui perantaraan program genetik.

Kemudian Tarakanov dan Prokaev (2006), mengajukan sebuah metode baru untuk mengidentifikasi CA, yaitu dengan menggunakan metode *immunocomputing*. Inti dari pendekatan ini adalah semacam perwakilan dari kondisi *(state)* dan transisi *automaton* dengan menggunakan jaringan kebal yang formal *(formal immune network)* yang dengan keakuratannya *(faultless)* mereduksi sejumlah transisi oleh proses imunisasi.

Sedangkan Vollmar (2005), berpendapat bahwa pada periode awal ilmu komputer, John von Neumann menyebarkan ilmu ini tidak hanya melalui paper seminar di dalam komputer-komputer tapi juga dengan memperkenalkan atau menggunakan CA yang terkoneksi dengan *self-reproduction* (sistem produksi mandiri).

Lainnya hal dengan Ardiyana dan Adityarani (2005), menjelaskan bahwa Algoritma Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. *Backtracking*, yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkina solusi yang ada.

Selanjutnya Primanio (2007), menjelaskan bahwa Labirin (*maze*) adalah permainan yang sudah tidak asing di telinga kita. Labirin adalah jaringan jalan yang rumit dan berliku-liku. Sejak zaman dahulu, labirin telah digunakan dalam berbagai kepentingan, mulai dari proteksi keamanan hingga hiburan. Pada umumnya, labirin dibuat untuk tujuan hiburan.

Dari beberapa jurnal yang sudah pernah di atas, hanya menggunakan satu algoritma saja, sedangkan algoritma tersebut masih ada beberapa kekurangan. Untuk itu penulis membandingkan dua algoritma yang sudah sering digunakan untuk mengetahui algoritma apa yang labih cocok untuk diterapkan kedalam suatu masalah.

Metode Penelitian

Selain metode literatur atau studi pustaka, RAD (*Rapid Application Development*) yang merupakan salah satu model proses *(process model)* yang ada dalam pengembangan sistem merupakan proses model yang digunakan dalam penelitian ini yang skema nya dapat dilihat pada Gambar 5.0 yang terlampir di bawah.

Model Pengembangan RAD memiliki empat fase yaitu:

1) Fase Menentukan syarat-syarat, yaitu menentukan tujuan dan syarat-syarat informasi.
2) Fase Perancangan, yaitu perancangan proses-proses yang akan terjadi dalam sistem dan perancangan antarmuka.
3) Fase Kontruksi, pada tahapan ini dilakukan tahap pengkodean terhadap rancangan-rancangan yang telah didefinisikan.
4) Fase Pelaksanaan, pada tahapan ini dilakukan pengujian terhadap sistem dan melakukan pengenalan terhadap sistem.

Pembahasan

Maze

Labirin (*maze*) adalah permainan yang sudah tidak asing di telinga kita. Labirin adalah jaringan jalan yang rumit dan berliku-liku. Sejak zaman dahulu, labirin telah digunakan dalam berbagai kepentingan, mulai dari proteksi keamanan hingga hiburan. Pada umumnya pembuatan labirin hanya untuk hiburan belaka. Namun, banyak bangunan yang menerapkan labirin sebagai salah satu sistem keamanan agar orang yang tidak berkepentingan atau tidak dikenal sulit untuk masuk ke dalam bangunan.
Labirin terbagi menjadi beberapa kategori sesuai jenisnya, yaitu Labirin 2 dimensi, 3 dimensi, bentuk segitiga, sigma, dan masih banyak lagi. Sebagai batasan materi, kita hanya akan membahas tentang pencarian jalan dalam labirin 2 dimensi yang gambarnya dapat lihat pada Gambar 5.1. yang terlampir dibawah.

Algoritma CA Dan *Backtracking*

Untuk CA, algoritma yang digunakan sama dengan fungsi transisi yang dimilikinya, yaitu :

Cellular Automata {
do *simultaneously* untuk semua sel {
if *state* sel tersebut == jalan && dikelilingi == 3 *neighbours* dinding {
ubah *state* sel tersebut menjadi *state* dinding
} until tidak ada lagi perubahan *state*
}}
masalah pencarian jalan ini diselesaikan dengan pendekatan *dead-end filler.* Dengan pendekatan ini, pencarian dilakukan dari sudut pandang *maze* (*maze* dilihat dari atas). Ide pendekatan ini adalah

Mulai dari salah satu *entry / exit point.*
do, *scan maze,*
for semua *dead-end* yang ditemukan,
tutup jalan dari *dead-end* hingga *junction* terdekat yang berhasil ditemukan.
While tidak ada jalan yang bisa ditutup lagi.

Sedangkan algoritma *backtrack* yang digunakan adalah sebagai berikut :
Backtrack {
Mulai pada salah satu *entry/exit point*
While (not end) { If (menemui junction),
Pilih salah satu jalan, lanjutkan perjalanan hingga menemui *dead-end* atau *entry/exit point* yang lain.
Else if (menemui *dead-end*),
Kembali ke *junction* terakhir
Else,
Jalan berdasarkan arah tertentu sampai menemui *dead-end* atau *entry/exit point* yang lain }}

Pada *backtrack,* pencarian dilakukan dari sudut pandang orang yang berada di dalam *maze.* Ide pendekatan ini adalah
Mulai dari salah satu *entry / exit point.*
do, if menentukan *junction,*
simpan posisi *junction,* pilih salah satu jalan, teruskan perjalanan (kembali ke do).
else if menemukan *dead-end,*
kembali ke posisi *junction* terakhir, pilih jalan yang belum dipilih,
teruskan perjalanan (kembali ke do).
else, teruskan perjalanan (kembali ke do).
until *entry / exit point* lain ditemukan.

Flowchart dari kedua algoritma di atas dapat dilihat pada Gambar 5.2 dan Gambar 5.3 yang terlampir dibawah.

Perancangan Proses

Dalam merancang proses pada simulasi pencarian jalan pada *maze* ini penulis menggunakan notasi UML sebagai *case tool* dalam merancang proses yang akan terjadi di dalam aplikasi, yakni dengan membuat *use case diagram*, *class diagram* dan *sequence diagram.*

Use Case Diagram

Pada implementasi ini, hanya ada satu *use case* yang digunakan yaitu *Use Case Input. Use case diagram* untuk implementasi ini dapat dilihat pada Gambar 5.4. di bawah dan untuk spesifikasi *use case* dapat dilihat pada tabel 5.1 di bawah.

Sequence Diagram

Sequence diagram berikut ini dibuat berdasarkan *use case diagram* yang telah dirancang sebelumnya. Adapun *sequence diagram* yang dibuat adalah *Sequence Diagram Input.* Pada *sequence diagram* ini dijelaskan bagaimana agar *user* dapat menentukan input yang diinginkan. Langkah dalam *sequence diagram* ini dapat dilihat Gambar 5.5 yang terlampir dibawah.

Class Diagram

Dalam Implementasi *Maze*-Ku ini terdapat satu buah kelas utama yaitu kelas **'MazeInterface'** dan satu kelas abstrak yaitu kelas **'Maze'**. kelas **BackTrack** yang merupakan kelas konkret dari kelas abstrak *maze* yang khusus menangani penyelesaian masalah *maze* dengan metode *backtrack* dan kelas **DeadEndFiller** yang merupakan kelas konkret dari kelas abstrak *maze* yang khusus menangani penyelesaian masalah *maze* dengan metode CA. Sedangkan kelas **Coordinate** merupakan kelas bantu untuk menyimpan data posisi suatu lokasi *maze.* Hubungan dari kelas ini merupakan hubungan asosiasi *(association). Class diagram* dari aplikasi ini Gambar 5.6. yang terlampir dibawah.

Perancangan Desain

a) Input

Input kedua program (metode *backtrack* dan CA) sama, yaitu *file teks* yang menggambarkan *maze input.* Satu *file input* hanya berisi satu *maze.* Didalam program, *maze* tersebut direpresentasikan dalam array 2-dimensi. Dengan dinding = 1 dan jalan = 0. contoh *file* input *maze*, serta representasi *maze* dalam program dapat dilihat pada gambar 5.7. dan 5.8. di bawah.

b) Output

Output kedua program (metode *backtrack* dan CA) sama, yaitu file teks. Satu *file* terdiri atas satu *maze.*

contoh file output yang inputnya seperti Gambar 5.7. di atas adalah Gambar 5.9. dibawah.

c) Strukutur Data

Struktur data untuk merepresentasikan *maze,* baik pada metode *backtrack* maupun CA, adalah *array 2*-dimensi. *Backtrack* membutuhkan tambahan memori untuk menyimpan posisi *junction* yang telah dilewati. Struktur data untuk menyimpan posisi *junction* tersebut adalah *stack (Last in firs out).* Struktur data digunakan karena ketika *backtracking* dilakukan (menemui *dead-end)*, maka program akan kembali ke posisi *junction* terakhir yang telah di laluinya.

Pada CA, tidak dibutuhkan tambahan memori berupa *stack,* namun CA membutuhkan tambahan memori berupa sebuah *array* 2-dimensi yang besarnya sama dengan besar *maze. Array* ini dibutuhkan sebagai *buffer* yang menyimpan konfigurasi CA pada *time-step* sebelumnya. *Array* 2-dimensi yang digunakan pada *backtrack,* bertipe data *byte.* Pemilihan tipe data ini disebabkan karena tiap sel *maze* pada *backtrack* mempunyai empat nilai (untuk menandakan dinding, jalan yang belum dilalui, jalan yang telah dilalui, dan jalan antara pasangan *entry* atau *exit point* yang telah diperoleh). Ini berarti lebih besar dari tipe data *Boolean* yang hanya mampu menyimpan dua nilai saja.

Fase Konstruksi

Maze yang digunakan berupa sebuah *file* teks yang dipanggil oleh program utama dari kelas utama yaitu **'MazeInterface'.** Adapun untuk hasil *output* program tersebut berupa *file* teks yang dihasilkan oleh *input* dengan memanggil suatu fungsi kelas yaitu **'Backtrack dan DeadEndFiller'.**Sedangkan untuk hasil *log* juga berupa sebuah *file* teks yang dihasilkan dari *output.* Hasil dari semua *output* dan *log* diimplementasikan kedalam tabel dengan menggunakan *microsoft excel* dan menghasilkan grafik perbandingan antara *backtrack dan deadend-filler.* Yang tabelnya dapat dilihat pada table 5.2. di bawah.

Fase Pelaksanaan

Pengujian Tampilan Implementasi untuk Pencarian Jalan Pada *Maze*

Percobaan yang dilakukan pada implementasi ini menggunakan ukuran 17x17, 33x33, 65x65, 129x129, 257x257, dan 513x513. *Maze* ini hanya mempunyai dua celah awal atau akhir yang terletak pada sisi terluar dari *maze.* Serta *maze* dengan jumlah pasangan *entry* atau *exit point* diatas penulis menggunakan ukuran 17x17, 33x33, dan 65x65. *Maze* dengan jumlah pasangan *entry* atau *exit point*-nya adalah 2, 4, dan 8.

Waktu Eksekusi

1) Maze dengan sepasang *entry* atau *exit point.*

Pada Gambar 5.10. (a) dibawah, terlihat bahwa urutan waktu (dari kecil ke besar) yang dibutuhkan oleh *backtrack* untuk memperoleh jalan tergantung pada banyaknya *junction* dan *dead-end.* Semakin banyak *junction,* semakin lama waktu yang dibutuhkan, tetapi semakin banyaknya *dea-end* maka semakin sedikit waktu yang dibutuhkan. Dalam hal ini, *maze* dengan *junction* terkecil mempunyai *running time* terbaik. Sebab semakin sedikit jumlah *junction* dalam suatu *maze* berarti semakin sedikit percabangan yang ada. Hal ini berarti pula, semakin sedikit *backtracking* yang harus dilakukan, sebab pilihan jalan yang ada juga semakin sedikit.

penyumbang waktu terbesar pada CA adalah proses *scanning* sel. Hal ini berarti, jika jumlah proses *scanning* ini bertambah (jumlah iterasi bertambah), total waktu eksekusi juga bertambah secara signifikan. Jumlah iterasi yang lebih besar berarti waktu eksekusi juga lebih besar. Hal ini terlihat pada Gambar 5.10. (b) di bawah. Pada Gambar 5.11. dibawah terlihat bahwa jumlah iterasi yang lebih besar berarti pula waktu eksekusi yang lebih besar. Jumlah iterasi bergantung pada persentase *dead-end* dan topologi *maze* itu sendiri.

Perbandingan antara total waktu eksekusi CA dengan *backtrack* dapat dilihat pada Gambar 5.12 Di bawah. Dari gambar, dapat dilihat bahwa, semakin besar ukuran *maze*, semakin besar pula bagian dari total waktu yang digunakan oleh CA. selain itu semakin besar ukuran *maze,* semakin tidak signifikan pula waktu yang dibutuhkan oleh *backtrack* untuk menyelesaikan masalah *maze* tersebut.

2) *Maze* dengan beberapa pasang *entry* atau *exit point*.

   Pada Gambar 5.13. (a) di bawah terlihat bahwa pada *backtrack,* penambahan jumlah pasangan *entry/exit point* menyebabkan penambahan waktu eksekusi yang konstan. Hal ini disebabkan karena pada *backtrack,* penambahan pasangan *entry* atau *exit point* berarti telah berhasil memperoleh satu jalan, *backtrack* harus dijalankan kembali untuk mencari jalan yang menghubungkan pasangan *entry* atau *exit point* yang lain.

   Sedangkan pada CA, hal ini tidak perlu dilakukan Karena pencarian jalan dilakukan dengan menutup semua jalan yang menuju *dead-end.* hal ini dapat dilihat pada Gambar 5.13. (b) Dibawah, penambahan jumlah pasangan *entry/exit point* menyebabkan penurunan waktu eksekusi.

   Untuk backtrack penambahan entry menyebabkan jalan yang dipilih menjadi lebih pendek. Sedangkan Penambahan jumlah *entry* tidak menambah jumlah *dead-end*

   Jika dilihat dari sumbangan masing-masing pendekatan terhadap waktu total, maka semakin besar jumlah pasangan *entry* atau *exit point,* semakin besar pula sumbangan dari pendekatan *backtrack.* Namun sumbangan dari CA semakin sedikit, lihat pada Gambar 5.14. di bawah. Hal ini disebabkan karena semakin banyak jumlah pasangan *entry* atau *exit point,* semakin lama pula waktu yang dibutuhkan oleh *backtrack* untuk menemukan jalan-jalan yang menghubungkan tiap-tiap pasangan *entry* atau *exit point* tersebut. Sebaliknya, semakin banyak pasangan *entry* atau *exit point,* waktu yang dibutuhkan oleh CA untuk menyelesaikan masalah tersebut, makin singkat atau setidaknya sama. Sebab pendekatan ini tidak memerlukan tambahan *cost* untuk mencari jalan-jalan antar *entry* atau *exit point* yang ada.

3) Jalan yang Ditemukan

   Sesuai dengan algoritmanya, untuk suatu pasang *entry* atau *exit point, backtrack* hanya dapat menemui satu jalan, sedangkan CA dapat menemui semua jalan yang menghubungkan *entry* atau *exit point* tersebut. Hal ini disebabkan karena pada CA menutup semua jalan yang menuju ke *dead-end,* sedangkan *backtrack* mencoba-coba jalan mana yang akan menuju ke *entry* atau *exit point.*

Kesimpulan

Metode *backtracking* dan metode CA untuk pancarian jalan pada *maze* dimplementasikan dengan menggunakan ukuran *maze* yang berbeda-beda, yang mempunyai dua celah awal atau akhir yang terletak pada sisi terluar dari *maze.* Perubahan waktu eksekusi yang dibutuhkan untuk memperoleh jalan yang menghubungkan dua buah *entry* atau *exit point* terhadap ukuran *maze*, oleh *backtrack* untuk memperoleh jalan tergantung pada banyaknya *junction* dan *dead-end* sedangkan CA sama dengan jumlah iterasi dikalikan waktu yang dibutuhkan untuk *scanning* seluruh sel per iterasi. Jumlah iterasi dan jumlah *walk* yang lebih besar berarti pula waktu eksekusi yang lebih besar, tetapi persentase *dead-end* yang semakin besar dapat menurunkan jumlah iterasi, yang berarti juga menurunkan lama waktu eksekusi. Pada *backtrack,* penambahan jumlah pasangan *entry* atau *exit point* menyebabkan penambahan waktu eksekusi yang konstan. Sedangkan pada CA, hal ini tidak perlu dilakukan karena pencarian jalan

dilakukan dengan menutup semua jalan yang menuju *dead-end.* Jumlah pasangan *entry* atau *exit point* yang semakin besar mempengaruhi jumlah iterasi menjadi lebih sedikit

Daftar Pustaka

Ardiyana dan Adityarani, "Analisis Penerapan Algoritma *Backtracking* Dalam Pencarian Solusi *Game* "Mummy Maze Deluxe", 2005

Fowler, Martin, *"UML Distilled",* Edisi ke-3. Penerbit Andi, Yogyakarta, 2004

Gardner, Martin. *"Mathematical Games The Fantastic Combinations of John Conway's New Solitaire Game Life",* Scientific American 223, October 1970

Hariyanto, Bambang, "Teori Bahasa, Otomata, dan Komputasi Serta Terapannya". Penerbit : Informatika, Bandung, 2004.

Hermawan, Benny, "Menguasai Java 2 dan *Object Oriented Programming",* Penerbit : Andi, Yogyakarta, 2004.

Hornby, A S. *"Oxford Advanced Learner's Dictionary Of Current English",* Fifth Edition, Oxford University Press, Oxford,1995.

John, Rajeev dan Jeffrey, "Toeri Bahasa Automata", Edisi 2, Penerbit Andi, Yogyakarta, 2001.

Limanto, Susana dan Anton Muljono. "Algoritma dan Pemrograman". Penerbit: Dinastindo, Jakarta, 2002.

Munir, Rinaldi, "Buku Teks Ilmu Komputer Algoritma dan Pemrograman dalam Bahasa Pascal dan C", Edisi 2, Penerbit Informatika, Bandung, 2002.

Nasuhi, Hamid dkk, "Pedoman Penulisan Karya Ilmiah", Cetakan pertama, Penerbit : CeQDA, Jakarta, 2007.

Nazir, Moh, "Metode Penelitian", Penerbit: Ghalia Indonesia, Jakarta, 985.

Preiss, Bruno, *"Abstract Backtracking Solvers",* 1997.

Pressman, Roger S, "Rekayasa Perangkat Lunak", Edisi 1, Penerbit: Andi, Yogyakarta, 2002.

Primanio, "Pencarian Jalan Keluar Labirin Dengan Metode *Wall Follower*, 2007.

Pullen, Walter D, *"Think Labyrinth",* Maze Algorithms, 2001.

Rucker, Rudy, *"Getting Started with Cellular Automata",* Cellular Automata Laboratory, 1990.

Sarkar, Palash, *"A Brief History of Cellular Automata",* ACM Computing Surveys, Vol 32, No. 1, March 2000.

Schatten, Alexander, *"Cellular Automata",* Digital World, 1999.

Sediyono, Eko, "Teknik Kompilasi Teori dan Praktek". Penerbit : Yogyakarta, 2005.

Slamet, Sumantri dan Heru Suhartanto,"Teknik Kompulasi", Penerbit: Elex Media Komputindo, Jakarta, 1992.

Suhendar, A dan Hariman Gunadi, "Visual Modeling Menggunakan UML dan Rational Rose", Penerbit: Informatika, Bandung, 2002.

Suryadi,MT. "Pengantar Analisis Algoritma", Edisi Pertama Cetakan ke-5, Penerbit : Gunadarma, Jakarta, 1996.

Tarakanov dan Prokaev, "Pada jurnal Mengidentifikasi Cellular Automata dengan immunocomputin", 2006.

Toffoli, Tomasso, Norman Margolus, *"Cellular Automata Machine: A New Environment*

for Modelling", The MIT Press, Cambridge Massachusetts, 1991.

Vollmar, "John von Neumann dan *Cellular Automata self-reproducing",* 2005.

Weimar, Joe R, *"Simulation with Cellular Automata Lecture Notes",* Technical University Braunschweig Institute of Scientific Computation, 1996.

Yacoubi dan Jacewicz, "Pendekatan Program Genetik untuk Identifikasi Struktur *Cellular Automata*", 2006.

http://www.astrolog.org/labyrnth/algorithm.html. Diakses pada 26 Desember 2007.

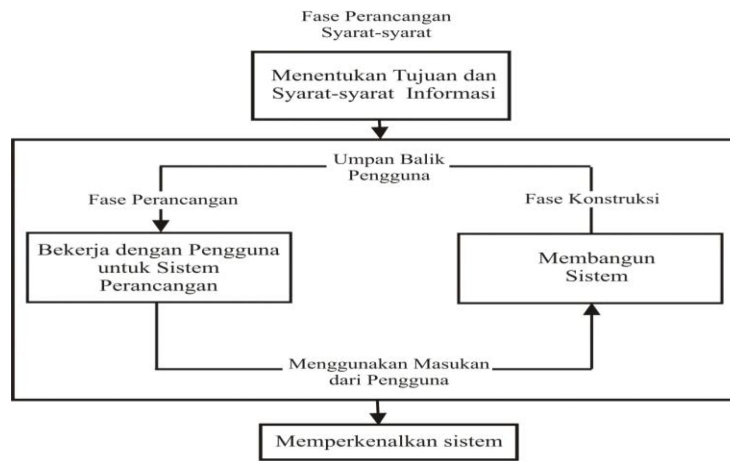
http://www.brpreiss.com/books. diakses pada 5 Januari 2008

http://www.ifs.tuwien.ac.at/~aschatt/info/ca/ca.html. diakses pada 23 September 2007

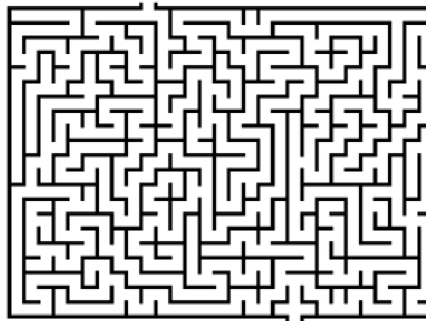
http://www.mathcs.sjsu.edu/faculty/rucker/celdoc/chap1.html. diakses pada 19 September 2007

http://www.tubs.de/institute/WiR/weimar/ZAscript. diakses pada 31 Oktober 2007

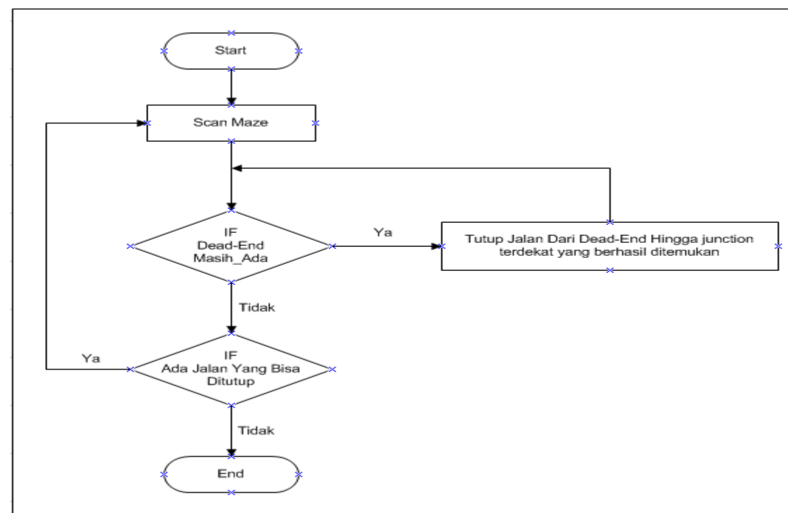
Lampiran Tabel / Gambar



Gambar 5.0
Skema Pengembangan Sistem RAD



Gambar 5.1.
Contoh *Maze* Labirin 2 Dimensi.



Gambar 5.2.
flowchart untuk CA dengan pendekatan *dead-end-filler*

Gambar 5.3.
flowchart untuk backtrack dengan pendekatan backtrack

```
17  17
# # # # # # # # # # # # @ # # # #
# # #                         # #
# # #   # # # # #   # # # #   # #
# # #   # # # # #   # # # #   # #
# # # # # # # # # # # # # #   # #
# #                           # #
# #         # # #   # # #     # #
# #   # #   # # #   # # #     # #
# #   # #   # # # # # # #     # #
# #   # #           # # #     # #
# # # # #   #   #   # # #     # #
# #         #   #   # # #     # #
# # # # #   #   #   # # #     # #
# # # # #   # # # # # # # # # # #
# #                         # # #
# #     # # # # # # #     # # # #
# # # # # # # # # # # # @ # # # #
```

Gambar 5.4.
File *Input Maze*

```
17  17
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1
1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 1 1 1 0 1 1 1 0 0 1 1
1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1
1 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1
1 1 0 1 1 0 0 0 0 0 1 1 1 0 0 1 1
1 1 1 1 1 0 1 0 1 0 1 1 1 0 0 1 1
1 1 0 0 0 0 1 0 1 0 1 1 1 0 0 1 1
1 1 1 1 1 0 1 0 1 0 1 1 1 0 0 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
```

Gambar 5.5.
Representasi *Maze* pada Program

```
17  17
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
```

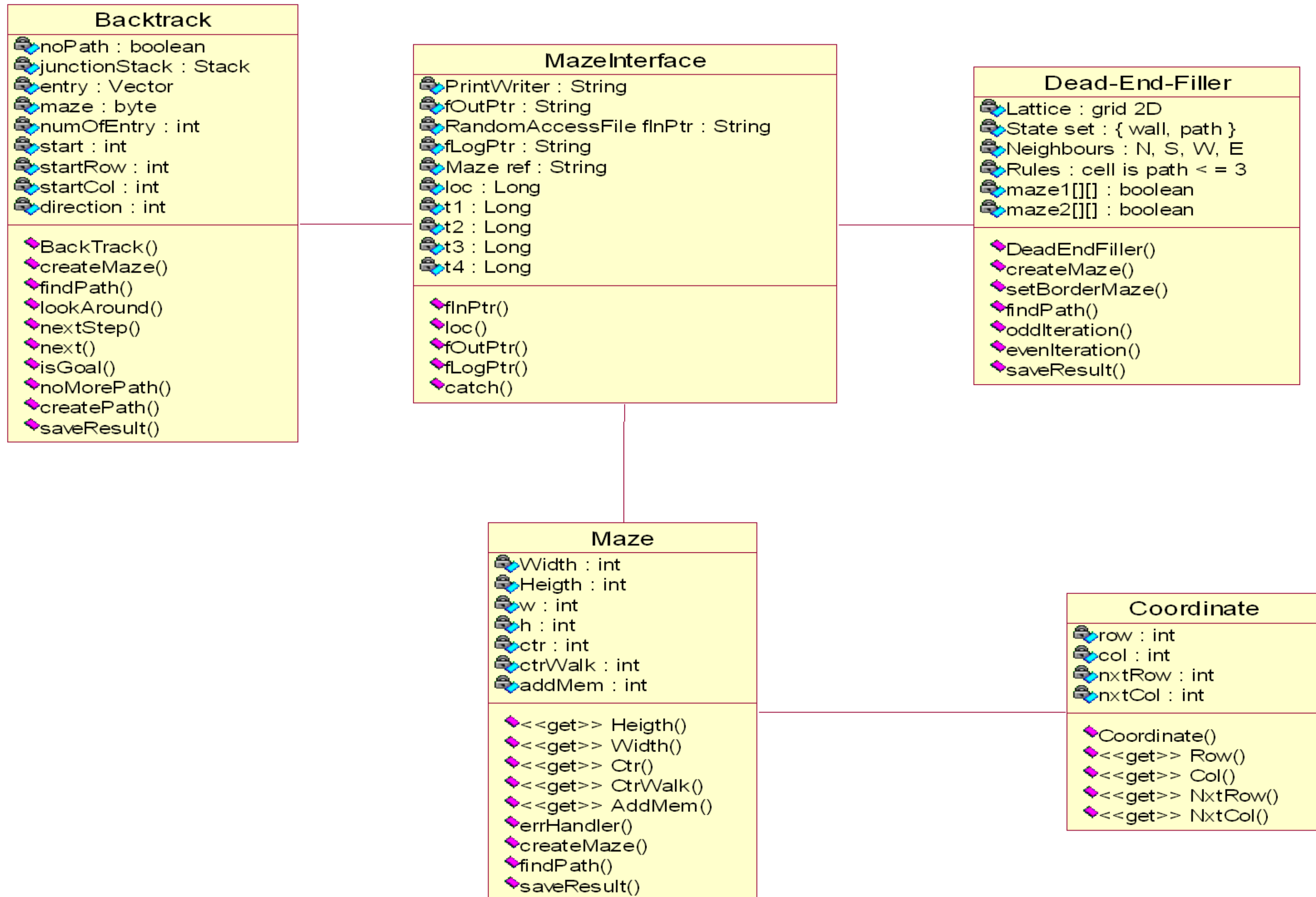
Gambar 5.6.
File *Output* untuk *input* gambar 4.4
(hasil CA = hasil *backtrack*)

User
Input

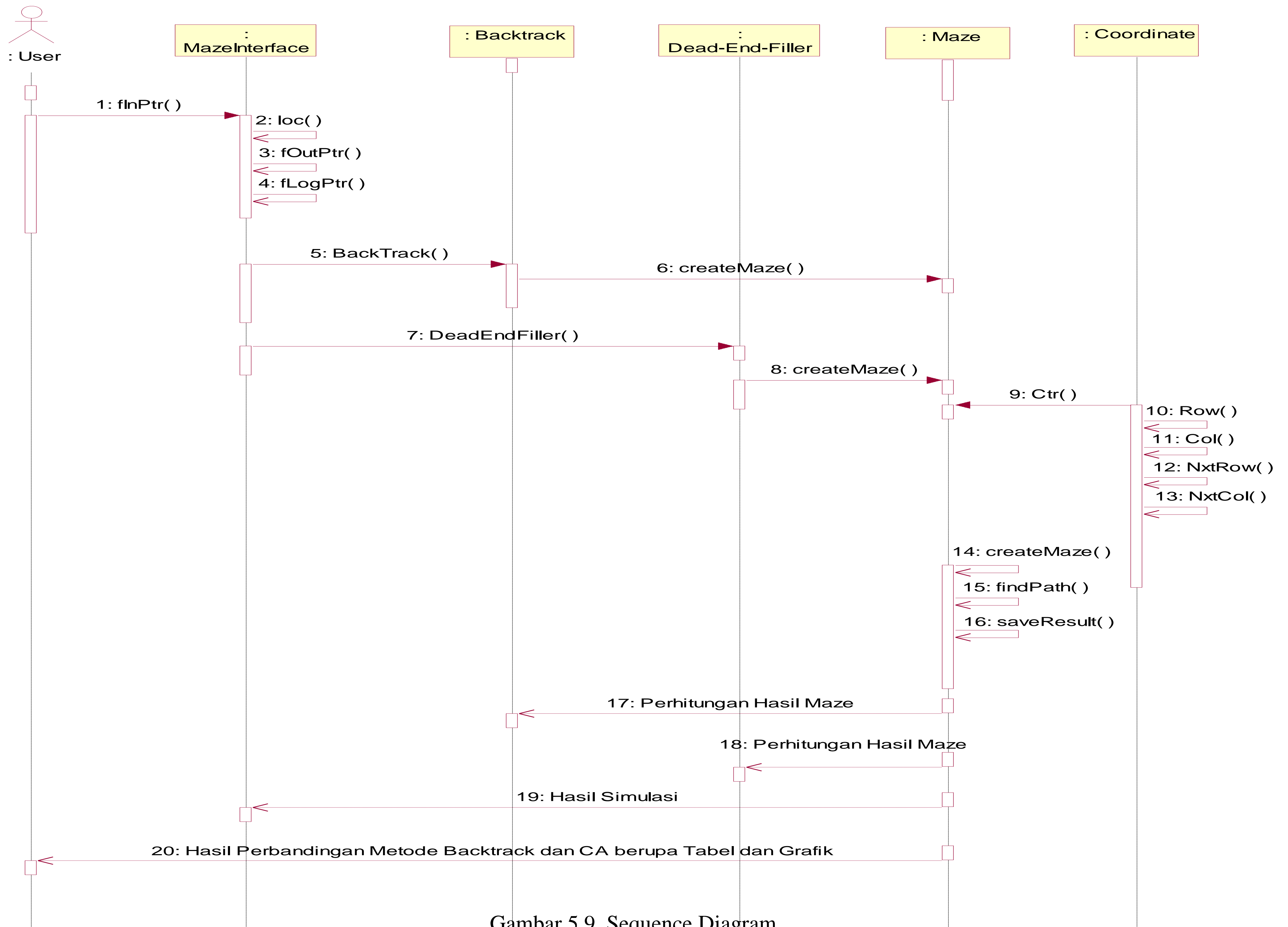
Gambar 5.7
Use Case Diagram

Use Case Name	*Input*
Actor	*User*
Brief Description	*Use case* ini digunakan untuk memasukkan input yang diinginkan oleh *user* berupa ukuran *maze* yang diinginkan yang telah tersedia dalam bentuk file.
Basic Flow	*1.* User memasukan input berupa ukuran *maze* 2. *Input* dari *user* digunakan untuk mengecek jalannya *maze* sampai menemukan jalan dan implementasi akan menampilkan hasil *output* dari CA dan *backtrack* serta *Log*.
Alternate Flow	1. Jika *user* memasukan *input* yang tidak sesuai dengan kelas utama pada aplikasi maka *user* akan menerima pesan kesalahan dan harus mengisi kembali input yang sesuai berupa *maze orthogonal.* 2. Apabila perubahan input yang ditampilkan oleh *user* gagal maka aplikasi akan menampilkan pesan *error*
Pre Condition	Buka implementasi java
Post Condition	File input yang dimasukkan disimpan di kelas utama

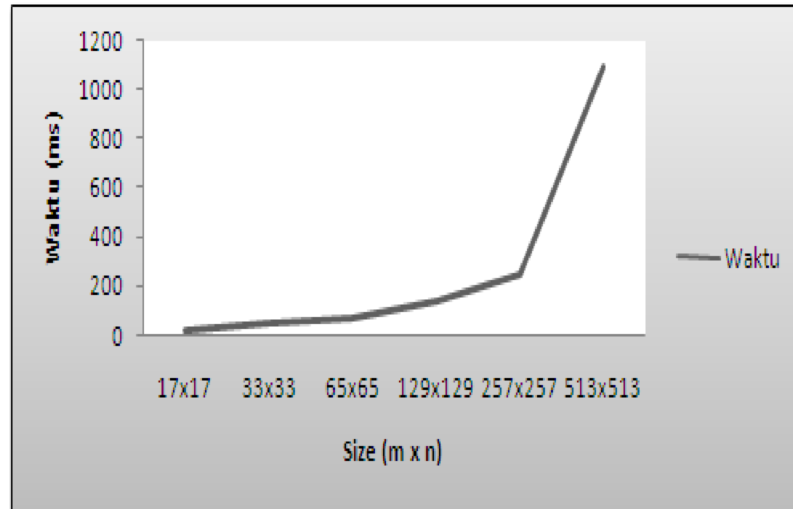
Table 5. 1
Spesifikasi *Use Case Input*



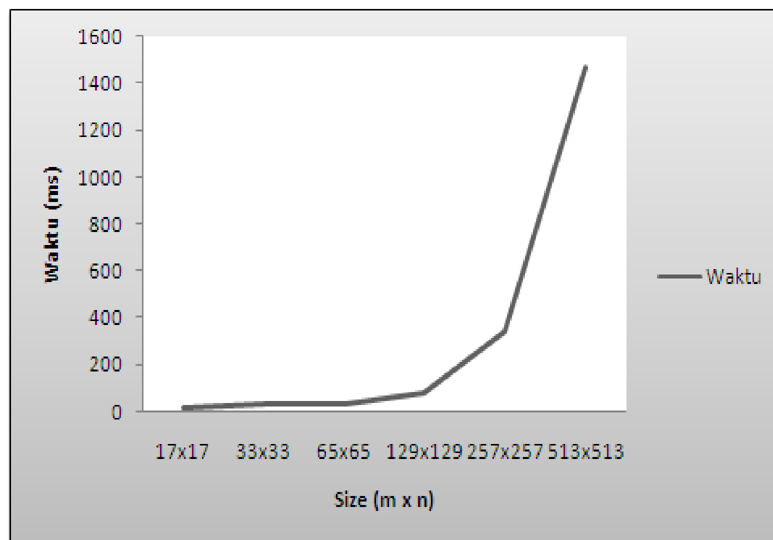
Gambar 5.8. *Class Diagram*



Gambar 5.9. Sequence Diagram

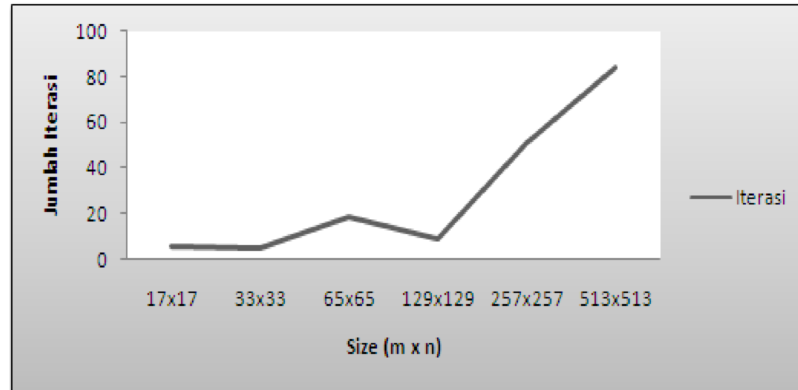


(a) *Backtrack*

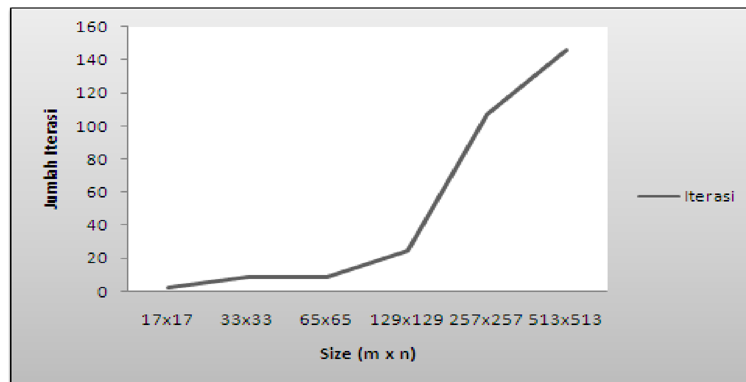


(b) CA

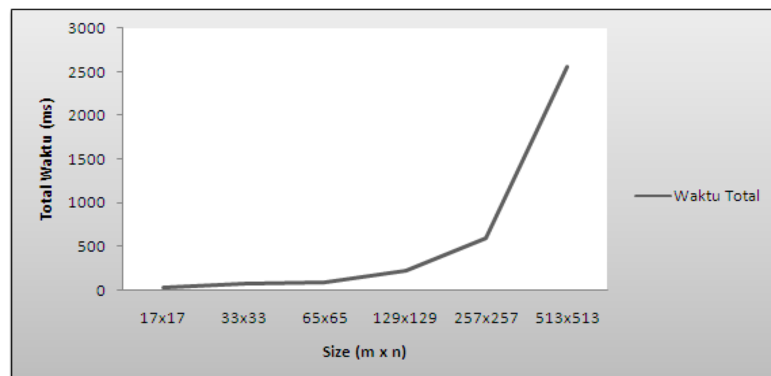
Gambar 5.10
Grafik Waktu Eksekusi Terhadap Ukuran *Maze* Dengan Sepasang *Entry* atau *Exit Point.*



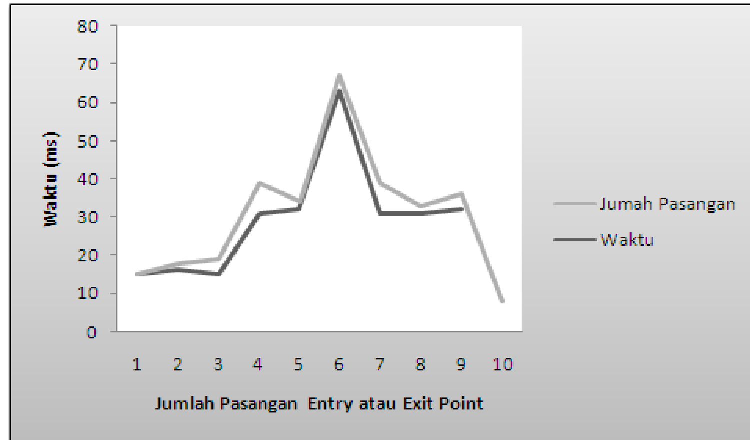
(a) Backtrack



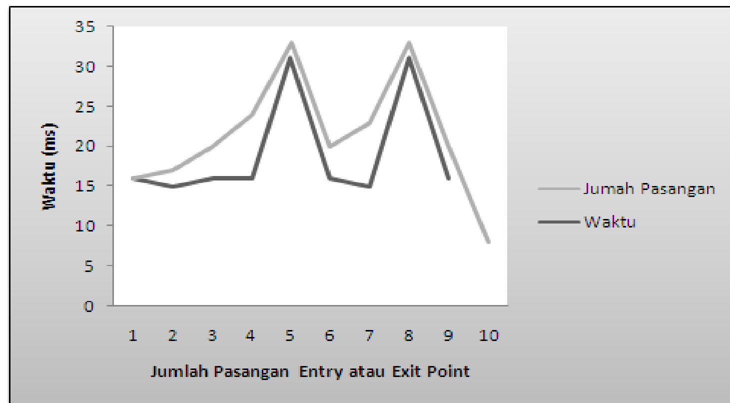
(b) CA
Gambar 5.11
Grafik Jumlah Iterasi Yang Dilakukan Terhadap Ukuran Maze



Gambar 5.12
Grafik Jumlah Total Waktu Terhadap Ukuran Maze

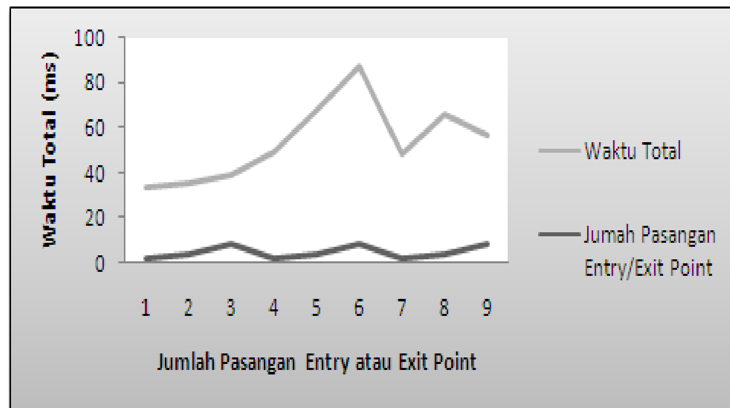


(a) Backtrack



(b) CA

Gambar 5.13
Grafik Waktu Eksekusi Terhadap Ukuran *Maze* dengan beberapa pasang *entry* atau *exit point.*



Gambar 5.14
Grafik Jumlah Total Waktu dengan beberapa pasang *entry* atau *exit point*