Implementasi Algoritma *Backtracking* dalam Pencarian Solusi *Flash Game Web-Based Maze*

Andre Novelando (13509085)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*13509085@std.stei.itb.ac.id*

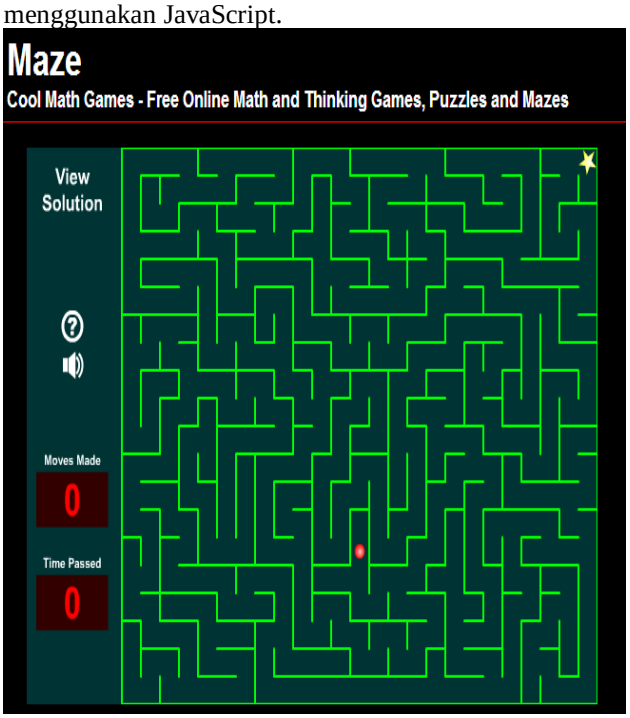
***Abstrak*—*Flash game* merupakan salah satu jenis *game* yang saat ini sangat populer. Di antara sekian banyak macam-macam *flash game, flash game web-based* merupakan yang paling banyak digemari. Dengan koleksi *game* yang banyak, menjadikan *flash game web-based* diminati oleh berbagai kalangan. Selain karena *user interface*-nya yang *friendly* dan *story* game-nya yang mudah dimainkan, *flash game web-based* bisa menjadi sarana *refreshing* di saat beraktivitas yang melelahkan dan berat. Pada makalah ini, *flash game web-based* yang dijadikan objek penelitian adalah *game Maze* karya *Cool Math Games*. *Maze* merupakan *game* sederhana yang bertujuan menentukan jalur yang tepat bagi sebuah bola untuk mencapai tujuan yang telah ditetapkan dari posisi awal yang sudah ditentukan. Navigasi bola dapat menggunakan tombol ← untuk arah ke kiri, tombol ↑ untuk arah ke atas, tombol → untuk arak ke kanan, dan tombol ↓ untuk arah ke bawah. *Maze* juga dapat menghitung jumlah pergerakan yang dilakukan dan waktu permainan. Pada makalah ini, dibahas tentang implementasi algoritma *backtracking* sebagai salah satu cara penyelesaian persoalan pada game ini. Pembangunan solusinya sendiri menggunakan JavaScript.**

***Kata Kunci*—*flash game web-based*, labirin, *Maze*, algortima *backtracking*, JavaScript.**

I. PENDAHULUAN

Hampir semua orang di dunia memainkan sebuah permainan (*game*). *Game* diminati karena dapat berfungsi ganda, yaitu sebagai *refreshing* dan olah otak. Banyaknya bentuk penyajian *game*, baik berupa *game* yang masih tradisional maupun yang modern yang didukung teknologi terkini, menjadikan *game* sangat populer. *Game* komputer merupakan salah satu *game* yang populer saat ini. Saat ini, terjadi migrasi besar-besaran ke arah aplikasi berbasis *web* dalam dunia komputer. Begitu juga yang terjadi pada *game* komputer. Saat ini, sangat banyak *game* yang berbasis *web*. Salah satu *game* berbasis *web* yang sangat populer, yaitu *flash game web-based*. Dalam makalah ini, *flash game web-based* yang dijadikan objek penelitian adalah *game Maze*. *Maze* merupakan permainan labirin yang diproduksi oleh *Cool Math Games*. *Maze* dapat dimainkan pada link berikut http://coolmath-games.com/0-maze/. *Maze* dimainkan dengan menjalankan atau menggerakkan sebuah bola menuju sebuah bintang dengan menggunakan tombol. Navigasi bola dapat menggunakan tombol ← untuk arah ke kiri, tombol ↑ untuk arah ke atas, tombol → untuk arak ke kanan, dan tombol ↓ untuk arah ke bawah. *Maze* juga dapat menghitung jumlah pergerakan yang dilakukan dan waktu permainan.

Pencarian solusi *Maze* menggunakan algoritma *backtracking*. Algoritma *backtracking* bertujuan untuk menentukan jalur yang sesuai dari posisi awal bola ke sebuah bintang. Selama proses pencarian jalur tersebut, jika menemui jalan buntu dan semua arah gerakan sudah dicoba, maka akan dilakukan proses *backtrack* sampai kembali menemukan jalur yang sesuai untuk mencapai sebuah bintang tersebut. Pembangunan solusi itu sendiri menggunakan JavaScript.



Gambar 1. *Game Maze*

II. STUDI LITERATUR

II.A. Labirin

Labirin merupakan permainan yang sudah sangat dikenal oleh smeua orang. Labirin memiliki struktur jalan yang rumit, penuh jalan buntu, dan hanya memiliki

beberapa pintu keluar atau pintu masuk. Labirin bisa dimanfaatkan untuk kepentingan keamanan maupun sebagai hiburan semata. Saat ini, labirin lebih dimanfaatkan untuk kepentingan hiburan saja. Tujuan permainan ini yaitu menemukan jalur dari sebuah pintu masuk ke pintu keluar atau suatu titik di dalam labirin yang sudah ditentukan.

Bentuk labirin dalam kehidupan sehari-hari yang mudah dijumpai yaitu struktur gang-gang kecil dan sempit di pemukiman penduduk yang padat. Seringkali seseorang yang baru berada di daerah tersebut tersesat di dalam gang-gang tersebut. Namun semua itu tidak perlu terjadi jika sudah mengetahui bagaimana caranya keluar dari labirin pemukiman padat tersebut. Banyak sekali cara yang bisa digunakan untuk keluar dari labirin tersebut. Adapun labirin yang sengaja dibangun untuk hiasan saja, seperti labirin dalam sebuah taman yang setiap dindingnya dibatasi oleh pohon, tembok, atau pagar. Labirin juga ada yang dibuat di atas kertas. Biasanya, merupakan pemodelan dari sebuah labirin. Labirin di atas kertas ini dapat dimainkan dengan menuliskan di atas kertas tersebut dengan pena atau pensil sebuah jalur dari pintu masuk atau posisi awal ke pintu keluar atau sebuah titik tertentu yang telah ditetapkan.

II.B. *Flash Game*

Flash pertama kali diperkenalkan pada tahun 1996. Saat pertama kali muncul di dunia ini, *flash* langsung menjadi sangat populer. Dengan pemanfaatan teknologi *flash,* dapat membangun *website, software* edukatif atau interaktif, animasi, *e-card,* iklan di media elektronik, *slide* presentasi, dan masih banyak hal yang bisa kita lakukan dengan pemanfaatan teknologi *flash. Tools* yang bisa digunakan untuk membuat sebuah aplikasi berbasis *flash,* yaitu Adobe Flash, SwishMax, Vecta 3D, Swift 3D, Amara, Kool Moves, dan banyak lagi aplikasi populer lainnya.

Flash game sudah menjadi sangat populer dan langsung mendapat tempat di hati masyarakat, khususnya *flash game web-bsed* karena dapat membuat menampilkan animasi dan *user interface* yang menarik di web. Dengan animasi dan *user interface* yang menarik, teknologi *flash* banyak didukung oleh berbagai pihak. Selain animasi dan *user interface* yang menarik, ukuran *file flash game web-based* yang kecil menjadikan *flash game web-based* mudah diakses oleh semua orang. Dengan begitu, kebutuhan *flash game web-based* akan teknologi *hardware* tidak berlebihan, yaitu cukup memiliki sebuah komputer yang sudah terpasang *browser* yang *support* untuk *flash.*

II.C. Algoritma *Backtracking*

Algoritma *backtracking* diperkenalkan pertama kali oleh D.H. Lehmer pada tahun 1950. Algoritma ini terus mengalami perkembangan. Sudah banyak perkembangan yang terjadi pada algoritma ini. Beberapa alhi yang mengembangkan algoritma *backtracking*, yaitu RJ Walker, Golomb, dan Baumert. Mereka membuat bentuk atau uraian umum tentang algoritma *backtracking.* Mereka juga berusaha mengimplementasikan algoritma *backtracking* dalam berbagai persoalan dan aplikasi di kehidupan sehari-hari.

Algioritma *backtracking* merupakan perbaikan dari algoritma *brute-force* (*exhaustive search*). Secara sistematis, algoritma *backtracking* mencari solusi persoalan di antara semua kemungkinan yang ada, tetapi hanya pencarian yang mengarah ke solusi saja yang dikembangkan, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi, sehingga waktu pencarian dapat dihemat dibandingkan dengan algoritma *brute-force*. Algoritma *backtracking* lebih tepat diformulasikan dalam bentuk algoritma rekursif.

Seperti yang sudah dikatakan sebelumnya bahwa algoritma *backtracking* melakukan pencarian pada sebuah ruang status, algoritma *backtracking* dapat dipandang sebagai sebuah metode pemecahan masalah yang terstruktur dan sistematis, dan algoritma *backtracking* juga dapat dipandang sebagai sebuah fase di dalam algoritma traversal *Depth-First Search* (DFS). Jika algoritma *backtracking* dipandang sebagai sebuah algortima berbasis DFS, pencarian solusi di dalam algoritma *backtracking* dapat dilakukan dengan menelusuri suatu struktur berbentuk pohon berakar secara *preorder*. Proses ini dicirikan dengan ekspansi simpul terdalam lebih dahulu sampai tidak ditemukan lagi suksesor dari suatu simpul.

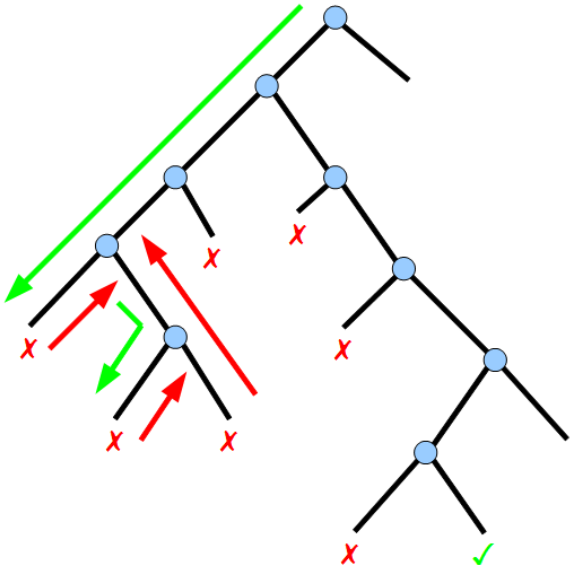
Properti umum algoritma *backtracking,* sebagai berikut:

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan *n-tuple*: $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$. Mungkin saja $S_1 = S_2 = \ldots = S_n$. Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1
2. Fungsi pembangkit
   Fungsi pembangkit untuk sebuah nilai x_k dinyatakan sebagai predikat $T(k)$. Fungsi $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
3. Fungsi pembatas
   Fungsi pembatas dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_k)$. B bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, x_k)$ dibuang.

Semua kemungkinan solusi disimpan di dalam sebuah ruang solusi (*solution space*). Ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai x_i. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai pohon ruang status (*state space tree*).

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

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan *depht-first search* (DFS).
2. Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*).
3. Simpul hidup yang *sedang* diperluas dinamakan simpul-E (*Expand-node*).
4. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
5. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*).
6. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*).
7. Simpul yang sudah mati tidak akan pernah diperluas lagi.
8. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul aras diatasnya
9. Lalu, teruskan dengan membangkitkan simpul anak yang lainnya.
10. Selanjutnya simpul ini menjadi simpul-E yang baru.
11. Pencarian dihentikan bila kita telah sampai pada *goal node*.



Gambar 2. Ilustrasi Algortima *Backtracking*

Skema umum algoritma *backtracking* (versi rekursif), yaitu:

```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan
metode runut-balik; skema rekursif
 Masukan: k, yaitu indeks komponen
vektor solusi, x[k]
 Keluaran: solusi x = (x[1], x[2], …,
x[n])
}
Algoritma:
   for tiap x[k] yang belum dicoba
sedemikian sehingga
        ( x[k]←T(k)) and B(x[1], x[2],
... ,x[k])= true do
      if (x[1], x[2], ... ,x[k]) adalah
lintasan dari akar ke daun
      then
         CetakSolusi(x)
      endif
      RunutBalikR(k+1)        { tentukan
nilai untuk x[k+1]}
   endfor
```

Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif. Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka untuk kasus terburuk, algoritma runut-balik membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

II.D. JavaScript

JavaScript merupakn sebuah bahasa skrip. Saat ini, JavaScript sangat populer di internet. JavaScript dapat bekerja di sebagian besar *browser* populer seperti Internet Explorer (IE), Mozilla Firefox, Netscape, Chrome, dan Opera. Penggunaan JavaScript yaitu dengan menyisipkan kode JavaScript ke dalam halaman web menggunakan tag SCRIPT. JavaScript berjalan pada sisi *client.* Ini berarti bahwa server bebas untuk mengerjakan sesuatu yang lain daripada pemrosesan instruksi untuk setiap klien. Hal ini telah membuat JavaScript pilihan yang lebih populer daripada bahasa yang memerlukan server untuk melakukan pengolahan.

javaScript dapat dimanfaatkan untuk:

1. Membuat teks berputar atau bergulir
2. Membuat bagian dari suatu bentuk terlihat atau tidak terlihat
3. Membuat *drop down menu*
4. Mengambil informasi tambahan dari *server* (Ajax) untuk sebagian me-*refresh* halaman
5. Memvalidasi input pengguna pada formulir
6. Melakukan perhitungan tanpa perlu kembali ke *server*
7. Menentukan jenis browser
8. Mengambil beberapa informasi dari jenis tertentu dari pengguna, seperti alamat *email* dari *bot*

Bahasa server-side seperti PHP atau Perl adalah cara terbaik untuk memodifikasi halaman sebelum loading, meskipun ada beberapa kasus Server-JavaScript (SJS). Tidak semua browser memiliki juru JavaScript (seperti browser hanya teks Lynx), atau menjalankan versi terbaru. Selain itu, beberapa pengguna mematikan kemampuan JavaScript dengan pilihan. Umumnya, halaman web harus menggunakan JavaScript untuk meningkatkan pengalaman pengguna, bukan bergantung padanya. Hal ini sering disebut sebagai degradasi anggun

(yaitu jika pengguna telah dimatikan JavaScript, halaman harus selalu masih beban, menyajikan informasi yang sama tetapi tanpa fungsi tambahan yang disediakan oleh JavaScript.)

Awalnya, JavaScript dikembangkan oleh Brendan Eich dari Netscape dibawah nama Mocha. Selanjutnya, nama Mocha diganti menjadi LiveScript. Penggantian namanya menjadi LiveScript dilatarbelakangi harapan agar bisa juga dimanfaatkan oleh *programmer* non-Java. Pada akhirnya, namanya menjadi JavaScript. Untuk diketahui, penamaannya menjadi JavaScript tidak ada hubungan sama sekali dengan bahasa Java. Bahkan mereka dikembangkan oleh dua perusahaan yang sama sekali berbeda, dengan tujuan dan pemikiran yang berbeda: Netscape mengembangkan JavaScript dan Sun Microsystems mengembangkan Java.

JavaScript dapat diinterpretasikan oleh *browser* yang paling langsung dan cepat, sementara Java memerlukan secara terpisah *Java Virtual Machine* harus dimulai sebelum menjalankan. JavaScript dan Java sama-sama menggunakan sintaks yang mirip (berdasarkan bahasa C) tetapi perintah yang digunakan banyak yang sangat berbeda. Ada juga perbedaan teknis. Java adalah bahasa diketik statis yang membutuhkan deklarasi semua variabel dan jenis mereka (misalnya integer, string atau boolean). Sebaliknya, Javascript adalah bahasa yang dinamis, memungkinkan variabel yang akan digunakan tanpa deklarasi sebelumnya.

III. SOLUSI PERSOALAN

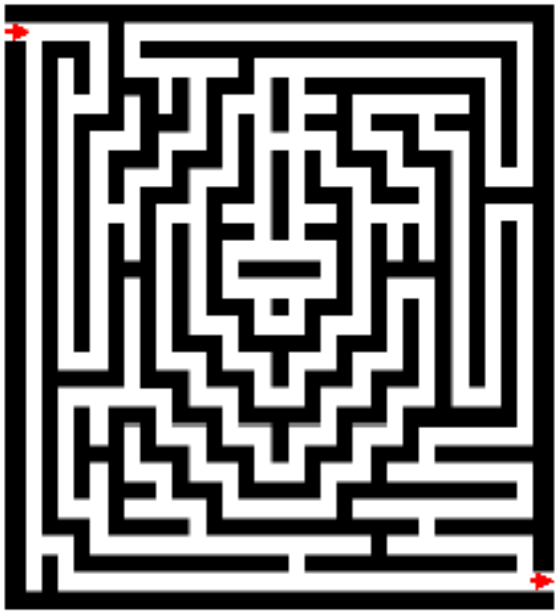
III.A. Implementasi Algoritma *Backtracking* pada *Game Maze*

Algoritma yang digunakan untuk mencari solusi persoalan labirin ini adalah algortima *backtracking.* Pada *game Maze*, algortima *backtracking* mencari jalur dari posisi awal ke sebuah bintang yang telah ditentukan oleh program permainan. Awalnya, algoritma ini mencoba sebuah lintasan. Jika dengan lintasan tersebut ditemui jalan yang buntu, akan dilakukan penandaan, lalu kembali ke langkah sebelumnya sampai ditemukan lintasan yang baru. Selanjutnya, lintasan tersebut juga ditelusuri. Begitulah seterusnya, sampai ditemukan lintasan yang mengarah ke posisi bintang, atau tidak terdapat solusi sama sekali setelah mencoba semua kemungkinan jalur yang ada.

Secara sistematis, penerapan algoritma *backtracking* untuk pencarian solusi *game Maze* dapat digambarkan sebagai berikut:

1. Bagi lintasan menjadi sederetan langkah
2. Sebuah langkah terdiri dari pergerakan satu unit sel pada arah tertentu
3. Arah yang mungkin: ke atas (up), ke bawah (down), ke kiri (left), ke kanan (right). Algoritma runut

Garis besar algortima backtracking-nya, yaitu:



Gambar 3. Contoh Labirin

```
while belum sampai pada tujuan do
  if terdapat arah yang benar sedemikian sehingga kita belum pernah
     berpindah ke sel pada arah tersebut
  then
     pindah satu langkah ke arah tersebut
  else
     backtrack langkah sampai terdapat arah seperti yang disebutkan
     di atas
  endif
endwhile
```

Lalu, pertanyaan selanjutnya adalah bagaimana mengetahui langkah yang mana yang perlu dijejaki kembali. Ada dua solusi untuk masalah ini, yaitu:

1. Menyimpan semua langkah yang pernah dilakukan
2. Menggunakan rekursi (yang secara implisit menyimpan semua langkah)

Dari kedua pilihan solusi di atas, rekursi adalah solusi yang lebih mudah.

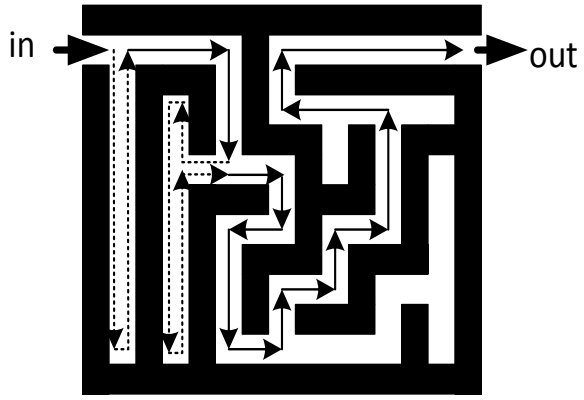
Algoritma *backtracking* untuk *game Maze* sebagai berikut:

```
function SolveMaze(input M : labirin)→boolean
{ true jika pilihan mengarah ke solusi }

Deklarasi
   arah : integer { up = 1, down, 2, left = 3, right = 4 }

Algoritma:
   if pilihan arah merupakan solusi then
```

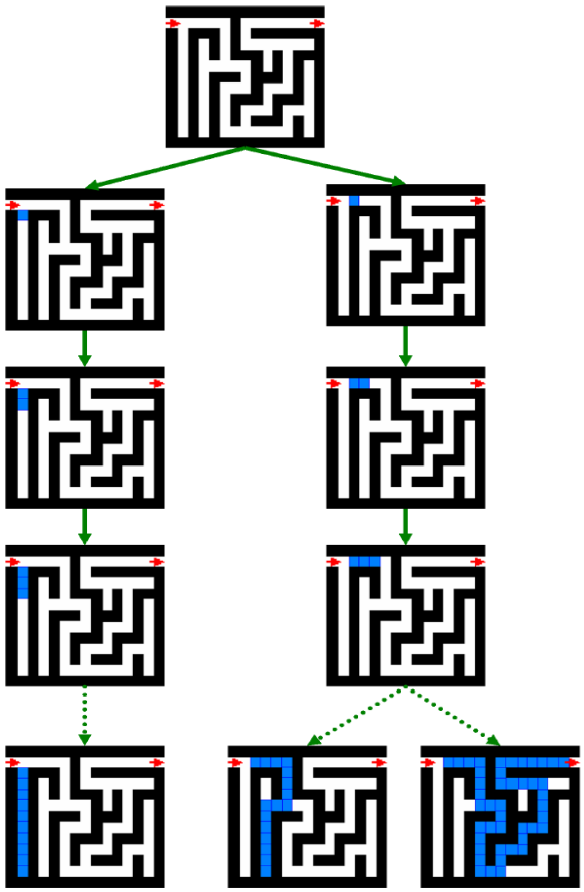
```
      return true
    else
      for tiap arah gerakan (lurus,
kiri, kanan) do
        move(M, arah)   { pindah satu
langkah (satu sel)
                          sesuai   arah
tersebut }
        if SolveMaze(M) then
          return true
        else
          unmove(M,    arah)        {
backtrack }
        endif
      endfor
      return false       { semua arah
sudah dicoba, tetapi
                          tetap   buntu,
maka
                          kesimpulannya:
bukan solusi }
    endif
```



Gambar 4. Ilustrasi Implementasi Algoritma *Backtracking* dalam Pencarian Solusi Labirin

Seperti yang sudah dibahas sebelumnya, algoritma *backtracking* melakukan pencarian solusi pada sebuah ruang solusi. Ruang solusi tersebut dapat digambarkan menjadi sebuah pohon ruang status. Akar dari pohon adalah posisi awal labirin, dan simpul-simpulnya adalah labirin yang dihasilkan dari pergerakan satu arah yang memungkinkan dari labirin sebelumnya atau labirin akarnya. Aturan pembentukan simpul-simpul menggunakan skema algoritma traversal DFS.

Simpul daun merepresentasikan titik *backtrack* atau simpul *goal*. Titik *backtrack* merepresentasikan bahwa pergerakan langkah menuju pada pada titik buntu. Simpul *goal* merepresentasikan bahwa pergerakan langkah menuju pintu keluar atau posisi yang sudah ditentukan. Jika pencarian solusi mencapai simpul *backtrack*, maka simpul daun tersebut dimatikan dan tidak diperluas atau ditelusuri lagi. Pencarian solusi dirunut-balik ke simpul-simpul di atasnya sampai ditemukan simpul yang mengarah ke solusi atau simpul *goal*. Demikian seterusnya, sampai solusi ditemukan atau tidak ada solusi.



Gambar 5. Sebagian Pohon Ruang Status Persoalan Gambar 4

III.B. Hasil Penelitian

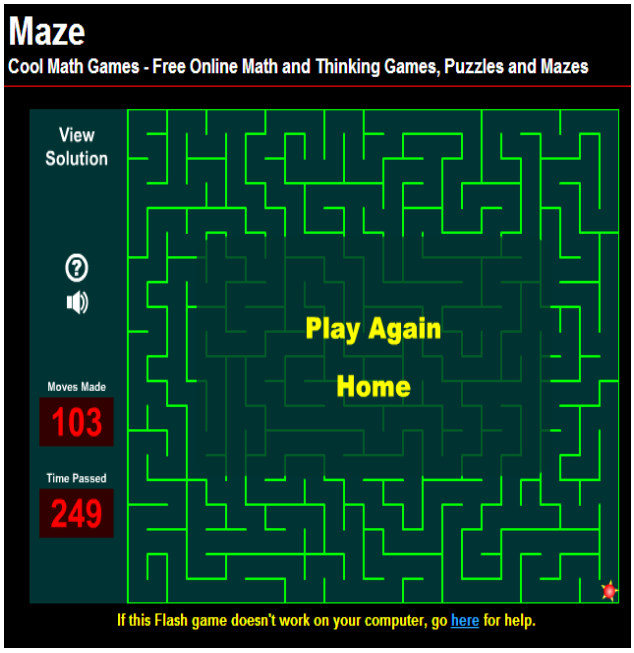
Implementasi solusi persoalan *game Maze* dibangun menggunakan JavaScript yang disisipkan ke dalam *browser*. Script di dalam JavaScript akan berjalan pada saat *browser* membuka *game Maze*. Seperti yang sudah di katakan sebelumnya, pembentukan simpul-simpul menggunakan skema algoritma traversal DFS. Prioritas pembangunan simpul berdasarkan arah gerak yang terurut dari yang paling diutamakan, yaitu: *atas*, *kiri*, *bawah*, dan *kanan*. Setelah melakukan pencarian dan menemukan solusi, aplikasi akan melakukan simulasi otomatis langsung pada *browser*. Dengan kata lain, aplikasi memainkan *game Maze* secara otomatis dengan jalur solusi yang sudah diperoleh sebelumnya

Berikut adalah hasil implementasi aplikasi:

1.Kasus 1

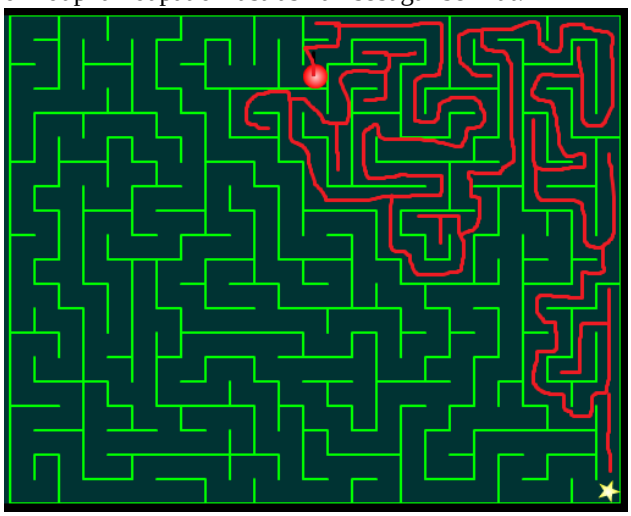
Gambar 6. Posisi Awal *Game Maze* pada Kasus 1

Dengan aplikasi, diperoleh jalur solusi dengan jumlah langkah yang dilakukan sebanyak 103 dan waktu eksekusi selama 249 detik.



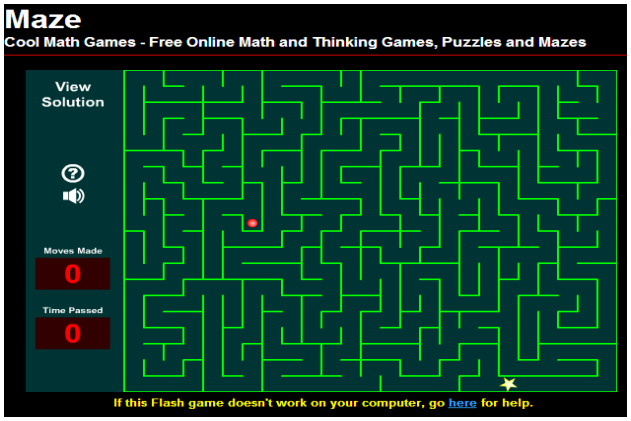
Gambar 7. Solusi *Game Maze* pada Kasus 1

Jalur yang dilalui bola dan simpul-simpul yang dihidupkan dapat diilustrasikan sebagai berikut:



Gambar 8. Ilustrasi Pencarian Solusi *Game Maze* pada Kasus 1

2.Kasus 2

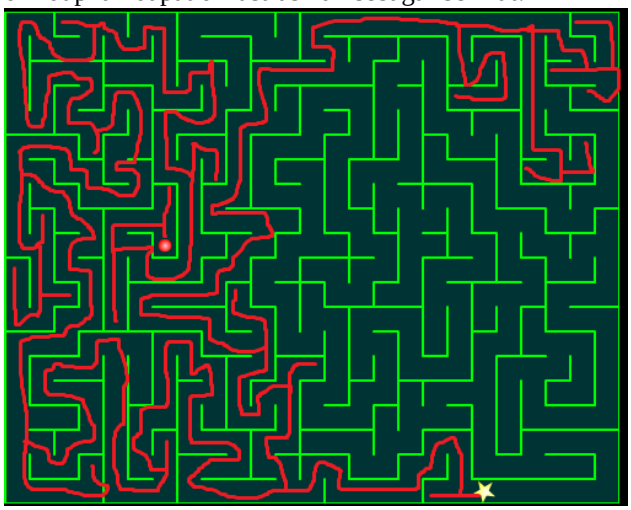


Gambar 9. Posisi Awal *Game Maze* pada Kasus 2

Dengan aplikasi, diperoleh jalur solusi dengan jumlah langkah yang dilakukan sebanyak 145 dan waktu eksekusi selama 318 detik.

Gambar 10. Solusi *Game Maze* pada Kasus 2

Jalur yang dilalui bola dan simpul-simpul yang dihidupkan dapat diilustrasikan sebagai berikut:



Gambar 11. Ilustrasi Pencarian Solusi *Game Maze* pada Kasus 2

III.C. Analisis

Dari hasil penelitian di atas, dapat diamati bahwa algoritma *backtracking* efektif dalam mencari solusi persoalan *game Maze*. Hal ini dikarenakan algoritma *backtracking* sangat sistematis dan hanya menelusuri simpul yang mengarah ke solusi. Kemampuan algortima ini dapat dilihat dari jumlah langkah yang dilakukan oleh suatu jalur solusi. Khsusu untuk waktu eksekusi aplikasi (dapat diliat dari waktu pada *game Maze*) sangat dipengaruhi oleh spesifikasi *hardware* komputer digunakan dan kecepatan jaringan internet digunakan. Dalam implementasi ini, faktor yang berperan sangat besar terhadap waktu mencapai solusi adalah kecepatan

jaringan internet, dimana digunakan koneksi internet dengan kecepatan yang lambat, sehingga waktu pencapaian solusi dapat dikatakan cukup lama.

IV. KESIMPULAN DAN SARAN

Dari hasil penelitian ini, dapat disimpulkan bahwa algoritma *backtracking* merupakan algoritmayang efektif untuk menyelesaikan persoalan labirin. Hal ini dikarenakan algortima *backtracking* hanya melakukan pencarian yang mengarah ke solusi sehingga tidak perlu menelusuri semua kemungkinan solusi yang ada. Dapat dikatakan, algoritma *backtracking* cukup efektif untuk persoalan atau *game* olah otak lainnya.

Sangat diharapkan penelitian selanjutnya bisa mengembangkan hasil penelitian ini menjadi sebuah solusi umum persoalan *game* labirin jenis *flash game web-based*, yang tidak hanya terbatas pada *game Math* karya *Cool Math Games* ini.

V. REFERENSI

[1] Hamilton, Naomi. 2008. *The A-Z of Programming Languages: JavaScript*. computerworld.com.au.

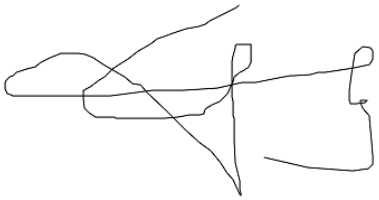
[2] Munir, Rinaldi. 2005. *Strategi Algoritmik*. Teknik Informatika ITB: Bandung

[3] Waldron, Rick . 2006. *The Flash History*. Flashmagazine

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.

Bandung, 19 Mei 2014

ttd



Andre Novelando (13509085)