Penerapan Algoritma BFS, DFS, dan IDS pada Penyelesaian Permainan *Bubble Blast 2*

Edward Samuel Pasaribu (13510065)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13510065@std.stei.itb.ac.id, [1]edward_samuel_esp@hotmail.com

Permainan Bubble Blast 2 merupakan salah satu permainan berjenis puzzle pada device dengan sistem operasi Android da iOS. Permainan ini menyediakan tantang berupa pemain harus menghilangkan semua gelembung yang ada pada papan permainan. Algoritma BFS, DFS, dan IDS dapat membantu untuk mencarikan solusi dari permainan ini. Dengan memanfaatkan algoritma BFS, DFS, dan IDS dilakukan pembangunan pohon ruang status secara dinamis dengan tujuan menemukan status tujuan yaitu saat papan permainan bersih dari gelembung-gelembung yang ada.

Index Terms*—BFS, *Bubble Blast 2*, DFS, IDS, *Solver

I. PENDAHULUAN

Permainan merupakan salah satu media hiburan yang banyak diminati. Permainan terdiri atas berbagai jenis, salah satunya adalah *puzzle*. Permainan jenis ini biasanya memberikan suatu permasalahan yang harus diselesaikan dengan strategi tertentu. Ada banyak sekali bentuk dari jenis permainan *puzzle* ini baik secara digital maupun fisik. *Bubble Blast 2* adalah satu dari sekian banyak jenis permainan *puzzle* pada *device* bersistem operasi Android dan iOS.

Sumber: https://lh4.ggpht.com/Xyjpj8cW5r-g9cOqTbeDJnS0aLOIZ2ek4lGV3rj2-C6e0BZiQdlRWvBQjFb4qD6GYw=w124

Gambar 1 Logo Bubble Blast 2 (versi Android)

Bubble Blast 2 dikembangkan oleh Magma Mobile. Pertama kali dirilis pada 3 Oktober 2010 pada lingkungan Android. Permainan ini termasuk permainan berjenis papan (*board game*). Pada permainan *Bubble Blast 2*, pemain disajikan sejumlah gelembung yang terdiri dari beberapa warna. Gelembung-gelembung ini harus dipecahkan semua dengan cara menyentuh gelembung tertentu sehingga papan permainan menjadi bersih dari gelembung.

Pada tulisan ini, akan dibahas bagaimana cara menemukan solusi permainan *Bubble Blast 2* dengan algoritma pencarian BFS, DFS, dan IDS sehingga didapatkan gelembung-gelembung mana saja yang harus disentuh.

Sumber: (kiri) http://a871.phobos.apple.com/us/r1000/086/Purple/v4/34/b8/44/34b84496-663a-4de6-ea3e-c3bb3390b9f3/mzl.mssveerf.320x480-75.jpg, (kanan) https://lh6.ggpht.com/WEFL5c_wuEZs8RNf9tU6jSeeklInCaryQrfjyukrwfmS-ql_s71raxlpQ9wZcdgJm_0=h230

Gambar 2 Permainan Bubble Blast 2 pada iOS (kiri) dan Android (kanan)

II. DASAR TEORI

A. Aturan Permainan Bubble Blast 2

Pada papan permainan *Bubble Blast 2* terdapat 4 warna gelembung sebagai berikut.

1. Gelembung merah, jika gelembung ini disentuh akan pecah (menghilang), lalu akan melepaskan 4 tembakan ke arah atas, kanan, bawah, dan kiri. Tembakan ini jika mengenai gelembung lain berakibat sama seperti gelembung tersebut disentuh.
2. Gelembung hijau, jika gelembung ini disentuh maka akan berubah menjadi gelembung merah.
3. Gelembung kuning, jika gelembung ini disentuh maka akan berubah menjadi gelembung hijau.
4. Gelembung biru, jika gelembung ini disentuh

maka akan berubah menjadi gelembung kuning.

Cara memainkan permainan ini adalah sebagai berikut.

1. Pemain akan disediakan papan yang terdiri dari satu atau beberapa gelembung
2. Sentuh beberapa gelembung satu per satu dengan maksimal sentuhan sesuai jumlah yang tertera pada layar.

Tujuan dari permainan ini adalah menghilangkan semua gelembung dengan maksimal jumlah sentuhan yang tertera pada layar.

B. Algoritma Pencarian Breadth-First Search

Algoritma *Breadth-First Search* (BFS) atau algoritma pencarian melebar merupakan salah satu algoritma untuk melakukan traversal pada graf. Algoritma ini menelusuri dari sebuah simpul akar pada graf lalu mengunjungi semua tetangga dari simpul tersebut.

Algoritma BFS memerlukan sebuah antrian (*queue*) untuk menyimpan simpul yang telah dikunjungi. Berikut ini adalah langkah-langkah traversal graf degan memanfaatkan algoritma BFS.

1. Masukan simpul akar ke dalam antrian.
2. Keluarkan sebuah simpul dari antrian.
3. Kunjungi semua tetangga dari simpul ini yang belum pernah dikunjungi sebelumnya. Masukan juga simpul-simpul tetangga ini ke dalam antrian.
4. Jika antrian telah kosong maka penelusuran telah selesai. Jika antrian masih ada, ulangi langkah nomor 2.

Berikut ini adalah *pseudocode* penelusuran graf dengan BFS. [1]

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.
  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke
layar
}
Deklarasi
   w : integer
   q : antrian

   procedure BuatAntrian(input/output q : antrian)
   { membuat antrian kosong, kepala(q) diisi 0 }
   procedure MasukAntrian(input/output q:antrian,
input v:integer)
   { memasukkan v ke dalam antrian q pada posisi
belakang }
   procedure HapusAntrian(input/output q:antrian,
output v:integer)
   { menghapus v dari kepala antrian q }
   function AntrianKosong(input q:antrian) → boolean
   { true jika antrian q kosong, false jika
sebaliknya }

Algoritma
   BuatAntrian(q) { buat antrian kosong }
   write(v) { cetak simpul awal yang dikunjungi }
   dikunjungi[v] ← true { simpul v telah dikunjungi,
tandai dengan true}
   MasukAntrian(q,v) { masukkan simpul awal
kunjungan ke dalam antrian}
   { kunjungi semua simpul graf selama antrian belum
kosong }
   while not AntrianKosong(q) do
      HapusAntrian(q,v) { simpul v telah dikunjungi,
hapus dari antrian }
      for tiap simpul w yang bertetangga dengan
simpul v do
         if not dikunjungi[w] then
               write(w) {cetak simpul yang
dikunjungi}
               MasukAntrian(q,w)
               dikunjungi[w] ← true
         endif
      endfor
   endwhile
   { AntrianKosong(q) }
```

C. Algoritma Pencarian Depth-First-Search

Algoritma Depth-First Search (DFS) atau algoritma pencarian mendalam juga merupakan salah satu algoritma untuk melakukan traversal pada graf. Algoritma ini menelusuri simpul tetangga pertama dari simpul yang dikunjungi saat ini, lalu masuk ke simpul anak dan melakukan hal yang sama sampai simpul tujuan ditemukan atau tidak ada lagi anak dari simpul tersebut. Jika hal itu terjadi, maka akan dilakukan runut balik (*backtrack*) ke simpul level atasnya.

Algoritma DFS secara alami menggunakan rekursif dalam implementasinya. Berikut ini adalah langkah-langkah traversal graf degan memanfaatkan algoritma DFS.

1. Kunjungi simpul akar.
2. Untuk sebuah tetangga dari simpul ini yang belum pernah dikunjungi sebelumnya, anggap simpul ini sebagai akar untuk melakukan DFS pada langkah 1.
3. Jika sudah tidak terdapat tetangga yang belum pernah dikunjungi lagi, lakukan runut balik ke simpul yang telah dikunjungi sebelumnya.
4. Penelusuran berhenti ketika simpul awal dari DFS ini sudah tidak mempunyai tetangga yang belum pernah dikunjungi.

Berikut ini adalah *pseudocode* penelusuran graf dengan DFS. [1]

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma
pencarian DFS
   Masukan: v adalah simpul awal kunjungan
   Keluaran: semua simpul yang dikunjungi ditulis ke
layar
}
Deklarasi
   w : integer

Algoritma
   write(v)
   dikunjungi[v] ← true
   for tiap simpul w yang bertetangga dengan simpul
v do
      if not dikunjungi[w] then
         DFS(w)
      endif
   endfor
```

C. Algoritma Pencarian Iterative-Deepening Search

Algoritma *Iterative-Deepening Search* (IDS) merupakan algoritma yang mengombinasikan keunggulan dari algoritma BFS dan DFS. Algoritma IDS bekerja seperti DFS dengan melakukan penelusuran secara mendalam terlebih dahulu tetapi dibatasi oleh nilai kedalaman. Ketika sudah mencapai simpul terdalam, maka penelusuran akan dirunut balik. Tetapi jika tidak ditemukan solusinya, maka akan dilakukan penambahan kedalaman. Pencarian akan berhenti ketika sebuah solusi ditemukan.

III. PENERAPAN ALGORITMA PENCARIAN

A. Strategi Pencarian Solusi

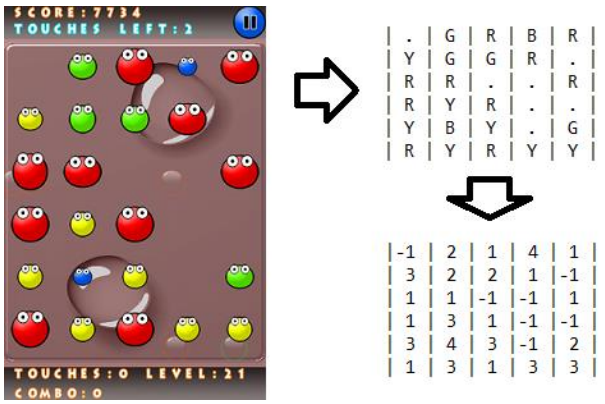
Pada permainan *Bubble Blast 2*, papan permainan bersifat dinamis. Oleh karena itu, penelusuran dengan memanfaatkan algoritma BFS, DFS, maupun IDS harus membuat pohon penelusuran secara dinamis juga dengan membuat pohon ruang status untuk papan permainan *Bubble Blast 2*.

Semua kondisi yang memungkinkan terjadi pada papan permainan disebut dengan **status persoalan**. Status persoalan ini dibentuk dengan "menyentuh" salah satu gelembung pada status persoalan yang tersedia. Status solusi dibentuk dari status-status persoalan yang merupakan solusi. **Status tujuan** dari permainan ini adalah saat semua gelembung pada papan permainan telah pecah. Simpul-simpul persoalan ini membentuk **pohon ruang status** dengan **simpul akar** adalah persoalan yang diberikan pada permainan.

Untuk itu status persoalan didefinisikan sebagai sebuah struktur data dengan larik 2 dimensi yang menyatakan papan permainan. Terdapat beberapa nilai yang memungkinkan mengisi larik 2 dimensi tersebut yaitu sebagai berikut tercantum pada Tabel 1.

Tabel 1 Nilai yang Memungkinkan pada Struktur Data Status Persoalan

Nilai	Keterangan
-1	Kondisi saat petak tidak terdapat gelembung
0	Kondisi saat petak terdapat gelembung yang baru pecah
1	Kondisi saat petak terdapat gelembung merah (R)
2	Kondisi saat petak terdapat gelembung hijau (G)
3	Kondisi saat petak terdapat gelembung kuning (Y)
4	Kondisi saat petak terdapat gelembung biru (B)



Gambar 3 Contoh Kasus Pemetaan Kondisi Permainan ke Struktur Data

Dengan demikian status tujuan dari permainan ini adalah mendapatkan nilai untuk setiap data adalah -1.

Sedangkan untuk membuat status persoalan tetangga/anak, diperlukan **langkah (*step*)** atau **operator perpindahan**. Dalam kasus permainan *Bubble Blast 2* ini, langkah yang digunakan adalah posisi gelembung yang disentuh, yaitu berupa nilai baris dan kolom sentuhan petak permainan.

Pada pencarian solusi ini, hanya akan dilakukan sampai ditemukan sebuah solusi. Pencarian tidak dilanjutkan dengan menelusuri keseluruhan status persoalan yang memungkinkan.

B. Penerapan Algoritma BFS

Pada pencarian solusi dengan algoritma BFS akan dibangkitkan sebuah simpul akar, yaitu kondisi papan permainan yang akan dicari solusinya. Lalu akan dibentuk simpul-simpul anak dengan "menyentuh" sebuah gelembung. Pencarian akan berhenti jika simpul yang sedang dikunjungi merupakan simpul solusi.

Berikut ini adalah *pseudocode* algoritma BFS untuk mencari solusi dari permainan *Bubble Blast 2*.

```
procedure BFS(input/output board:state)
Deklarasi
   finish : boolean
   current : state
   newstate : state
   queue : antrian
   visitedlist : list of state
   posisisentuhan : step

   procedure MasukList(input/output l:list of state,
input v:state)
   { memasukkan v ke dalam list l }
   procedure MasukAntrian(input/output q:antrian,
input v:state)
   { memasukkan v ke dalam antrian q pada posisi
belakang }
   procedure HapusAntrian(input/output
q:antrian,output v:state)
   { menghapus v dari kepala antrian q }
   procedure TambahAnak(input/output current:state,
input/output newstate:state)
   { Menambahkan newstate sebagai anak dari current
}
   procedure TandaiSolusi(input/output
newstate:state)
   { Menandai bahwa newstate merupakan solusi }
```

```
   function AntrianKosong(input q:antrian) → boolean
   { true jika antrian q kosong, false jika
sebaliknya }
   function Solusi(input s:state) → boolean
   { true jika s merupakan solusi dari permasalahan
}
   function Mengandung(input l:list, input
board:state) → boolean
   { true jika board terdapat dalam list l }

   function BuatStatusBaru(input current:state,
input posisisentuhan:step) → state
   { membuat status baru dengan "menyentuh"
gelembung status current pada posisisentuhan }

Algoritma
   MasukAntrian(queue, board)
   MasukList(visitedlist, board)

   finish ← false
   while (not AntrianKosong(queue) and not finish)
      HapusAntrian(queue, current)

      for setiap posisisentuhan gelembung yang dapat
disentuh do
         newstate ← BuatStatusBaru(current,
posisisentuhan)
         if (Solusi(newstate))
            finish = true
            TambahAnak(current, newstate)
            TandaiSolusi(newstate)
            return
         else
            if (not Mengandung(visitedlist,
newstate)) then
                TambahAnak(current, newstate)
                MasukAntrian(queue, newstate)
                MasukList(visitedlist, newstate)
            endif
         endif
      endfor
   endwhile
```

C. Penerapan Algoritma DFS

Pada pencarian solusi dengan algoritma DFS juga akan dibangkitkan sebuah simpul akar, yaitu kondisi papan permainan yang akan dicari solusinya. Lalu akan dibentuk simpul-simpul anak dengan "menyentuh" sebuah gelembung. Penelusuran akan dilakukan secara kontinu terhadap status yang telah dibuat pertama kali. Pencarian akan berhenti jika simpul yang sedang dikunjungi merupakan simpul solusi.

Berikut ini adalah *pseudocode* algoritma DFS untuk mencari solusi dari permainan *Bubble Blast 2*.

```
procedure DFS(input/output board:state, input/output
visitedlist:list of state)
Deklarasi
   newstate : state

   procedure MasukList(input/output l:array of
state, input v:state)
   { memasukkan v ke dalam list l }
   procedure TambahAnak(input/output current:state,
input/output newstate:state)
   { Menambahkan newstate sebagai anak dari current
}
   procedure TandaiSolusi(input/output
newstate:state)
   { Menandai bahwa newstate merupakan solusi }
   function Solusi(input s:state) → boolean
   { true jika s merupakan solusi dari permasalahan
}
   function Mengandung(input l:list, input
board:state) → boolean
   { true jika board terdapat dalam list l }

   function BuatStatusBaru(input current:state,
input posisisentuhan:step) → state
   { membuat status baru dengan "menyentuh"
gelembung status current pada posisisentuhan }

Algoritma
   if (Solusi(board))
      TandaiSolusi(board)
   else
      for setiap gelembung yang dapat disentuh do
         newstate ← BuatStatusBaru(current, posisi
sentuhan)
         if (not Mengandung(visitedlist, newstate))
then
            TambahAnak(current, newstate)
            MasukList(visitedlist, newstate)

            DFS(newstate, visitedlist)

            if (Solusi(newstate))
               return
            endif
         endif
      endfor
   endif
```

D. Penerapan Algoritma IDS

Pada pencarian solusi dengan algoritma IDS juga akan dibangkitkan sebuah simpul akar, yaitu kondisi papan permainan yang akan dicari solusinya. Lalu akan dibentuk simpul-simpul anak dengan "menyentuh" sebuah gelembung. Penelusuran akan dilakukan secara kontinu terhadap status yang telah dibuat pertama kali tetapi dengan batas yang ditentukan. Batas ini akan dimulai dari 1, dan akan terus dinaikkan hingga ditemukan solusi. Pencarian akan berhenti jika simpul yang sedang dikunjungi merupakan simpul solusi.

Berikut ini adalah *pseudocode* algoritma IDS untuk mencari solusi dari permainan *Bubble Blast 2*.

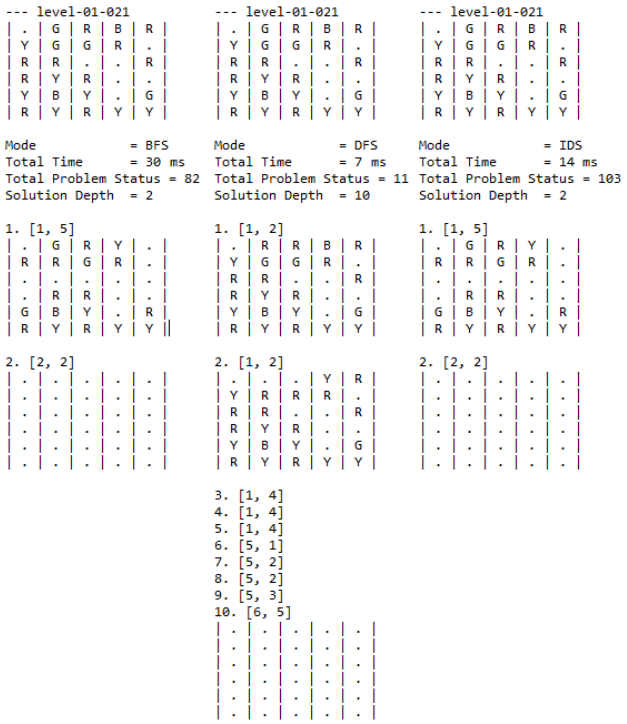
```
procedure IDS(input/output board:state)
Deklarasi
   solution : boolean
   depth : integer

   procedure KosongkanList(input l:list of state)
   { Mengosongkan list l }
   function Solusi(input s:state) --> boolean
   { true jika s merupakan solusi dari permasalahan
}
   procedure DLS(input/output board:state, input
depth:integer)
   { Pencarian dengan Depth-Limited Search }

Algoritma
   depth ← 0; solution ← false
   while (not solution)
      DLS(board, depth)
      if (Solusi(board)) then
         solution ← true
      else
         depth++
```

```
      endif
   endwhile
```

```
procedure DLS(input/output board:state, input
depth:integer)
Deklarasi
   newstate : state

   procedure TambahAnak(input/output current:state,
input/output newstate:state)
   { Menambahkan newstate sebagai anak dari current
}
   procedure TandaiSolusi(input/output
newstate:state)
   { Menandai bahwa newstate merupakan solusi }

   function Solusi(input s:state) → boolean
   { true jika s merupakan solusi dari permasalahan
}
   function BuatStatusBaru(input current:state,
input posisisentuhan:step) → state
   { membuat status baru dengan "menyentuh"
gelembung status current pada posisisentuhan }

Algoritma
   if (Solusi(board))
      TandaiSolusi(board)
   elseif (depth > 0)
      for setiap gelembung yang dapat disentuh do
         newstate ← BuatStatusBaru(current, posisi
sentuhan)
         TambahAnak(current, newstate)
         DLS(newstate, depth-1)
         if (Solusi(newstate))
            return
         endif
      endfor
   endif
```

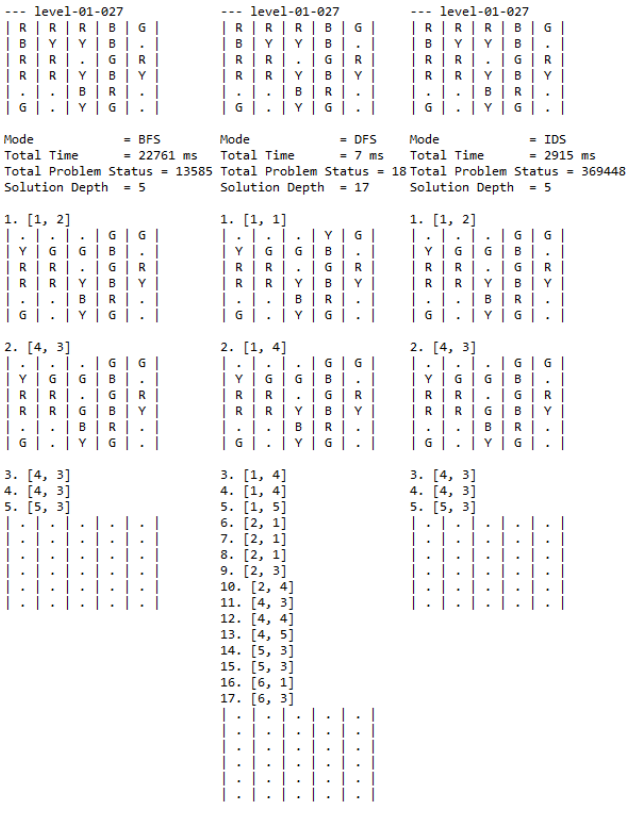
IV. HASIL PERCOBAAN DAN ANALISIS

Algoritma yang diberikan pada bagian III diimplementasikan dengan bahasa pemrograman Java (kode sumber terlampir). Berikut ini contoh hasil pengujian pada kasus permainan *pack* 1 *level* 21 pada Gbr. 4 dan *pack* 1 *level* 27 pada Gbr. 5. Pada hasil terdapat keterangan mode algoritma apa yang sedang digunakan, total waktu dalam milidetik, total status permasalahan yang dibuat, kedalaman solusi. Selain itu juga ditampilkan langkah-langkah apa saja yang diambil untuk mencapai solusi dalam format [X, Y] dengan X menyatakan baris gelembung dan Y menyatakan kolom gelembung mana yang harus disentuh.



Gambar 4 Hasil Pengujian dengan Algoritma BFS, DFS, dan IDS pada kasus permainan Pack 1 Level 21

Pada kasus permainan *pack* 1 *level* 21 permainan dibatasi maksimal jumlah sentuhan adalah 2 sentuhan. Algoritma BFS dan IDS berhasil menemukan solusi dengan jumlah sentuhan sebanyak 2 sentuhan. Tetapi jumlah status persoalan yang dibangkitkan berbeda, BFS membangkitkan 82 status sedangkan IDS membangkitkan 103 status. Solusi yang diberikan algoritma DFS tidak memenuhi jumlah maksimum langkah yang diperbolehkan. DFS menghasilkan 10 langkah untuk mencapai status tujuan.



Gambar 5 Hasil Pengujian dengan Algoritma BFS, DFS, dan IDS pada kasus permainan Pack 1 Level 27

Pada kasus permainan *pack* 1 *level* 27 permainan dibatasi maksimal jumlah sentuhan adalah 6 sentuhan. Baik BFS maupun IDS berhasil menemukan solusi dengan jumlah 5 solusi, kurang dari batas yang diperbolehkan. Status permasalahan yang dibangkitkan BFS jauh lebih sedikit dibandingkan IDS, tetapi pada IDS unggul dalam waktu penyelesaian. Sama seperti sebelumnya, DFS memberikan solusi dengan jumlah langkah yang sangat banyak, yaitu 17 langkah untuk mencapai solusi.

Pada algoritma BFS, pencarian solusi dilakukan secara melebar dengan membuat status permasalahan per kedalaman terlebih dahulu. Hai ini memungkinkan BFS untuk melakukan pencarian dengan solusi yang minimum. Tetapi dengan algoritma ini terdapat *cost* yang harus dibayar, yaitu harus memeriksa apakah setiap status sudah pernah diperiksa atau belum sehingga menyebabkan algoritma BFS menjadi jauh lebih lama karena terjadi pemeriksaan tersebut.

Pada algoritma DFS, pencarian solusi dilakukan selalu dengan "menyentuh" gelembung terkiri atas. Dengan demikian maka solusi yang selalu dihasilkan oleh DFS akan menjadi sangat panjang dibanding algoritma BFS maupun IDS.

Pada algoritma IDS, pencarian dilakukan dengan melakukan *Depth-Limited-Search*, yaitu algoritma DFS yang dibatasi oleh kedalaman tertentu. Pada algoritma ini, jika kedalaman dari solusi besar, maka status permasalahan yang muncul akan semakin besar. Harga yang harus dibayar oleh algoritma ini adalah untuk melakukan pengujian terhadap kedalaman yang tidak mempunyai solusi, dan terdapat komputasi ulang beberapa status permasalahan yang ada untuk setiap kedalaman yang berbeda. Walaupun demikian, baik BFS dan IDS sama-sama menghasilkan solusi yang optimum.

Pada permainan *Bubble Blast 2*, lebih diperlukan solusi dengan jumlah yang tidak melewati batas maksimum sentuhan terhadap gelembung. Untuk itu, jika dibandingkan dengan ketiga algoritma yang digunakan BFS dan IDS yang dapat memenuhi batasan ini dengan selalu menghasilkan solusi optimum yang tentunya tidak akan melebihi batas maksimum sentuhan yang diberikan. Tetapi, jumlah sentuhan rata-rata setiap papan permainan tidak terlalu banyak, maka algoritma yang terbaik di antara algoritma BFS, DFS, dan IDS untuk pencarian solusi permainan *Bubble Blast 2* adalah dengan menggunakan algoritma IDS karena waktu eksekusi yang jauh lebih singkat dibandingkan BFS.

V. SIMPULAN

Permainan *Bubble Blast 2* dapat dicari solusinya dengan memanfaatkan algoritma BFS dan IDS. Algoritma DFS tidak dapat dimanfaatkan karena hampir selalu menghasilkan solusi yang melebihi batas kedalaman yang diperbolehkan.

Algoritma BFS dan IDS dapat menghasilkan solusi yang optimum, sehingga tidak akan pernah melebihi batas sentuhan yang diizinkan. Algoritma BFS pada kebanyakan kasus bekerja lebih lambat dibandingkan IDS karena terdapat biaya tambahan untuk melakukan pengecekan apakah suatu status pernah terjadi atau belum. Kedalaman solusi pada permainan ini juga relatif sangat kecil, sehingga algoritma IDS cukup mangkus dalam pencarian solusi. Dengan demikian algoritma IDS menjadi pilihan terbaik di antara algoritma pencarian BFS, DFS, dan IDS untuk menemukan solusi dari permainan *Bubble Blast 2*.

VI. LAMPIRAN

Kode sumber yang dibuat pada tulisan ini dapat diunduh pada pranala http://code.google.com/p/bubble-blast-2-solver/.

DAFTAR REFERENSI

[1] Rinaldi Munir, Diktat Kuliah IF3051 Strategi Algoritma, Bandung: Program Studi Teknik Informatika ITB, 2009.

[2] https://itunes.apple.com/id/app/bubble-blast-2-!/id493274935?mt=8, diakses pada 16 Desember 2012 pukul 9.10 WIB.

[3] https://play.google.com/store/apps/details?id=com.magmamobile.game.BubbleBlast2&hl=en, diakses pada 16 Desember 2012 pukul 9.00 WIB.

[4] http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/2003-04/intelligent-search/breadth.html, diakses pada 16 Desember 2012 pukul 10.00 WIB.

[5] http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/2003-04/intelligent-search/depth.html, diakses pada 16 Desember 2012 pukul 10.20 WIB.

[6] http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/2003-

04/intelligent-search/inter.html, diakses pada 16 Desember 2012 pukul 10.30 WIB.

PERNYATAAN

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

Bandung, 16 Desember 2012

Edward Samuel Pasaribu
13510065