PENGGUNAAN ALGORITMA *BRUTE FORCE* DALAM PERMAINAN PERMAINAN *THE LEGEND OF DRUNKEN MAN*

Wiradeva Arif Kristawarman
13505053

Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl.Ganesa 10, Bandung Indonesia
e-mail: if15053@students.if.itb.ac.id

ABSTRAK

Saat ini algoritma *brute force* sudah jarang dipakai untuk kecerdasan buatan (*artificial intelligence*) dalam sebuah permainan (*games*) terlebih lagi dalam permainan yang sudah sangat kompleks baik dari segi grafis ataupun *artificial intelligence* seperti Winning Eleven 10, Final Fantaxy XII, atau permainan yang paling populer saat ini yaitu Warcraft III Frozen Throne DoTA.
Walaupun begitu, untuk program permainan sederhana algoritma *brute force* masih cukup mangkus digunakan terutama dari segi implementasi. Hal ini dikarenakan *game-game* kecil tersebut biasanya dibuat untuk kesenangan pembuatnya dan dibuat dalam waktu yang singkat sehingga mereka lebih memilih algoritma yang mudah diimplementasikan, seperti *brute force* untuk kecerdasan buatan daripada algoritma lain seperti *backtracking, IDA*,* atau *branch and bound* yang lebih sukar diimplementasikan.
Program permainan yang menggunakan algoritma *brute force* sebagai algoritma kecerdasan buatannya yang akan dibahas pada makalah ini adalah permainan *The Legend of DrunkenMan.* Permainan *The Legend of DrunkenMan* ini merupakan tugas dari Mata Kuliah IF2281 Pemrograman Berorientasi Objek di Informatika ITB yang dibuat secara kelompok dan salah satu anggotanya adalah penulis . Harapan penulis menulis makalah ini adalah agar memberikan pandangan kepada pembaca bahwa dengan algoritma yang sederhana kita bisa membuat kecerdasan buatan yang sudah cukup baik namun tetap mengingatkan pembaca untuk tetap mengembangkan algoritma yang lebih cerdas agar permainan menjadi lebih menarik.

Kata kunci: *brute force*, permainan, optimasi, kecerdasan buatan, *The Legend of DrunkenMan.*

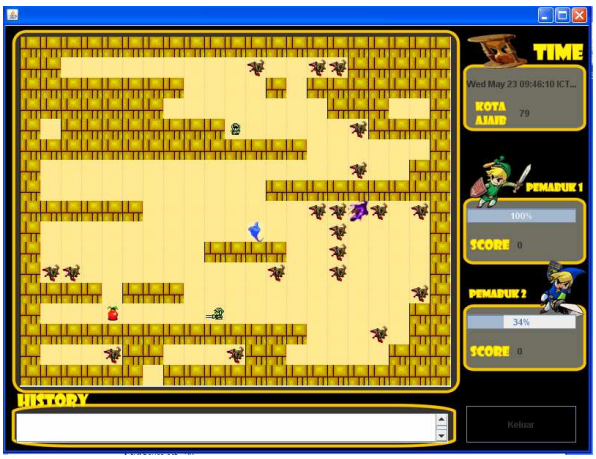
1. PENDAHULUAN

1.1. Deskripsi Masalah

Permainan *The Legend of DrunkenMan* merupakan permainan yang dikembangkan oleh kelompok CiWork++ dengan penulis sebagai salah satu anggotanya. Permainan ini pertama kali dibuat bertujuan untuk melengkapi tugas dalam mata kuliah pemrograman berorientasi objek. Oleh karena itu, elemen yang paling diutamakan dalam permainan ini adalah *Graphical User Interface* yang menari dan interaktif (syarat sebuah permainan), sedangkan *artificial intelligence* yang ada di permainan ini bukan menjadi elemen yang utama.

Permainan ini sendiri bercerita tentang seorang jagoan yang tinggal di sebuah kota ajaib. Orang ini dikenal dengan sebutan *DrunkenMan* oleh penduduk sekitar karena setiap malam selalu ada di bar yang ada di kota untuk minum-minum bersama teman-temannya yang lain. Penduduk di kota ajaib ini setiap malam selalu terganggu oleh keberadaan serigala dan monster yang banyak berkeliaran yang kemunculannya disebabkan oleh sesuatu yang misterius. Oleh karena itu, raja dari kota ajaib ini menyelenggarakan sayembara bagi siapa yang bisa mengembalikan ketentraman kota ajaib dengan cara membasmi serigala dan monster yang ada di kota akan mendapatkan harta yang berlimpah. Dan tentu saja jagoan kota *DrunkenMan* tidak ingin ketinggalan sayembara ini karena dia sangat membutuhkan uang untuk melanjutkan kebiasaannya minum-minum di bar. Dalam sayembara ini raja hanya menerima 2 orang paling sakti saja yang boleh mengikuti sayembara ini.

Permainan *The Legend of DrunkenMan* adalah game dengan tampilan 2 dimensi dengan sudut pandang kamera dari atas. Tampilannya sendiri diimplementasikan sebagai *matrix of PictureBox*. Setiap grid(titik koordinat yang ada di matriks) pada matriks itu bisa berupa tembok atau jalan yang melambangkan keadaan kota ajaib tersebut, atau bisa juga berupa pemabuk, serigala, monster, pedang, pedang super, buah, atau fly wing yang merupakan elemen game yang berinteraksi satu sama lain.



Gambar 1. Tampilan permainan The Legend of DrunkenMan

Karena raja hanya menyelenggarakan sayembara ini untuk dua kontestan saja, maka permainan ini hanya terdiri dari dua pemabuk sebagai tokoh utama. Untuk mode permainan dua pemain, maka kedua pemabuk digerakkan oleh dua orang pemain dan untuk mode permainan satu pemain, maka pemabuk yang pertama digerakkan oleh pemain sedangkan pemabuk yang kedua digerakkan oleh sistem pada komputer yang menggunakan diberikan kecerdasan buatan.

Permainan ini boleh dibilang sangat sederhana. Sehingga untuk menggerakkan pemabuk yang digerakkan oleh sistem, kami membuat algoritma *brute force* yang naif sebagai kecerdasan buatan dalam permainan ini.

Pergerakan pemabuk yang digerakkan oleh komputer secara garis besar memiliki kelakuan sebagai berikut :

a. Jika nyawanya masih banyak (melebihi batas yang ditetapkan, dalam hal ini 20 atau 40% dari nyawa maksimum pemabuk) dan pemabuk masih menggunakan tangan kosong, maka pemabuk akan berusaha mencari pedang yang ada dalam radius 3 kotak dari pemabuk (kota 7 x 7 dengan pemabuk ada di tengah). Namun jika tidak ada pedang dalam radius tersebut maka pemabuk akan bergerak menuju monster atau harimau yang paling dekat. Jika pemabuk sudah memiliki pedang pemabuk akan berusaha mencari item seperti buah atau sayap untuk teleport yang ada dalam radius 3 kota dari pemabuk (kotak 7 x 7 dengan pemabuk ada di tengah). Namun jika tidak ada item dalam radius tersebut, maka pemabuk akan bergerak menuju monster atau harimau yang paling dekat dengan pemabuk.
b. Jika nyawa pemabuk sudah kritis (kurang dari batas yang ditetapkan, dalam hal ini 20), pemabuk akan berusaha mencari peri untuk mendapatkan penyembuhan sehingga nyawa pemabuk menjadi maksimum lagi. Pencarian peri ini tidak mempedulikan keadaan pemabuk memakai senjata atau masih tangan kosong.

Kelakuan yang begitu mementingkan nyawa dibuat karena dalam permainan ini jika seorang pemabuk mati, maka pemabuk lainnya lah yang kemungkinan besar akan menjadi pemenang sayembara atau dengan kata lain menjadi pemenang permainan ini.

1.2. Dasar Teori

Metode pemecahan masalah yang paling sederhana dan paling mudah adalah algoritma *brute force*. Istilah lain yang bertalian dengan *brute force* adalah *exhaustive search. Brute force* adalah sebuah pendekatan yang lempeng (*straightforward*)untuk memecahkan suatu masalah, biasanya didasarkan langsung pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas(*obvious way*).

Algoritma *brute force* pada umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesainnya, terutama bila masalah yang dipecahkan berukuran besar(dalam hal ini ukuran masukannya). Kata "*force*" sendiri mengindikasikan kesan "tidak intelektual". Kadang-kadang algoritma *brute force* disebut juga algoritma naif (*naive algorithm*).

Algoritma *brute force* seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantuk kita menemukan algoritma yang lebih cerdas dan lebih mangkus.

Untuk masalah yang ukurannya kecil, kesederhanaan *brute force* biasanya lebih diperhitungkan daripada ketidakmangkusannya. Algoritma *brute force* sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang mangkus. Misalnya pada perhitungan a^n jika diselesaikan dengan algoritma *brute force* mudah diperlihatkan bahwa kompelksitas waktunya adalah dalam O(n). Bila perhitungan a^n diselesaikan dengan metode *Divide and Conquer*, maka kompleksitas waktunya jauh lebih baik daripada teknik *brute force* yaitu O(log n).

Meskipun *brute force* bukan merupakan teknik pemecahan masalah yang mangkus, namun metode *brute force* dapat diterapkan pada sebagian besar masalah. Agak sukar menunjukkan masalah yang tidak dapat dipecahkan dengna metode *brute force.* Bahkan ada masalah yang hanya dapat dipecahkan secara *brute force.* Beberapa pekerjaan mendasar di dalam komputer dilakukan secara *brute force*, seperti menghitung jumlah dari n buah bilangan, mencari elemen terbesar di dalam tabel, dan sebagainya. Selain itu, algoritma *brute force* seringkali lebih mudah diimplementasikan daripada algoritma yang lebih canggih, dan karena kesederhanaannya, kadang-

kadang algoritma *brute force* dapat lebih mangkus(dtinjau dari segi implementasi).

2. METODE

Untuk membuat algoritma *brute force* yang akan digunakan sebagai kecerdasan buatan dalam permainan ini akan dibuat beberapa prosedur dan fungsi. Prosedur yang paling utama dibuat terlebih dahulu yang akan memanggil prosedur-prosedur lain yang ada di bawahnya. Prosedur ini yang akan menentukan kapan pemabuk akan mencari monster dan harimau, peri, senjata, atau item sesuai dengan keadaan pemabuk saat itu. Berikut ini adalah *pseudo-code* dari prosedur tersebut :

```
procedure pemabukMelangkah(input
myKota:Kota, P:Pemabuk)
{Kota merupakan objek yang memiliki atribut
array dari makhluk-makhluk yang ada di kota
dan matriks of grid dengan tiap grid
menyatakan tembok,jalan,serigala, monster,
senjata, atau item-item lainnya}
Deklarasi
  temp:integer;

Algoritma
  if (P.nyawa>20) then
     If (P.weapon=tanganKosong) then
        temp=cariSenjata(myKota,P)
     else
         temp=cariItem(myKota,P)
     endif
     if (temp=0) then
        {jika tidak ada item atau pedang
dalam radius 3 kotak maka akan mencari
musuh}
        temp=cariMusuh(myKota,P)
     endif
  else
     temp = cariPeri(myKota,P)
  endif
  P.arah=temp
  P.maju
```

Jadi ada 4 fungsi yang salah satunya akan dipanggil oleh prosedur pemabukMelangkah yaitu cariSenjata, cariItem,cariMusuh, cariPeri. Keempat fungsi ini akan mengembalikan integer yang merepresentasikan arah yang harus dituju jika pemabuk akan melangkah maju. Jika nyawa pemabuk misalnya 40 dan sedang tidak memakai senjata, maka yang akan dilakukan adalah mencari senjata dalam radius 3 kotak, jika tidak ada (temp=0) maka dia akan mencari musuh yang paling dekat dari posisinya sekarang kemudian melangkah sesuai arah yang didapat. Jika nyawa pemabuk misalnya 40 dan sedang memakai pedang, maka yang akan dilakukan adalah mencari item dalam radius 3 kotak. Jika tidak ada, maka dia akan mencari musuh. Jika nyawa pemabuk misalnya 11 dan memakai pedang, maka dia akan tetap melangkah mendekati peri.

Fungsi cariSenjata sendiri mempunyai *pseudo-code* seperti berikut :

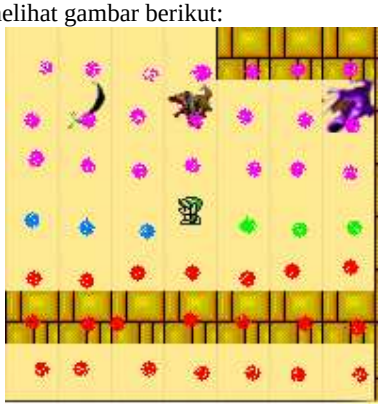
```
function cariSenjata(input myKota:Kota,
P:Pemabuk)
{Kota merupakan objek yang memiliki atribut
array dari makhluk-makhluk yang ada di kota
dan matriks of grid dengan tiap grid
menyatakan tembok,jalan,serigala, monster,
senjata, atau item-item lainnya
Keluaran adalah integer 1,2,3,4, atau 0. 1
untuk utara,2 timur,3 selatan,4 barat, 0
untuk tidak ada arah yang cocok}
Deklarasi
  i,j:integer;

Algoritma
  for i=P.x-3 to P.x do
     if
(myKota.grid[i,P.y]=pedang)or(myKota.grid[i
,P.y]=pedangSuper) then
        →4
     endif
  endfor
  for i=P.x+3 downto P.x do
     if
(myKota.grid[i,P.y]=pedang)or(myKota.grid[i
,P.y]=pedangSuper) then
        →2
     endif
  endfor
  for i=P.x-3 to P.x+3 do
     for j=P.y-3 to P.y do
        if
(myKota.grid[i,j]=pedang)or(myKota.grid[i,j
]=pedangSuper) then
           →1
        endif
     endfor
  endfor
  for i=P.x-3 to P.x+3 do
     for j=P.y+3 downto P.y do
        if
(myKota.grid[i,j]=pedang)or(myKota.grid[i,j
]=pedangSuper) then
           →3
        endif
     endfor
  endfor
  →0
```

Maksud dari algoritma ini adalah, pada loop pertama dia memeriksa apakah ada senjata di 3 kotak sebelah barat(kiri) dari pemabuk jika ada dia akan melangkah ke kiri(arah 4). Jika tidak ada berpindah ke loop berikutnya, algoritma akan memeriksa apakah ada senjata di 3 kotak sebelah timur(kanan) dari pemabuk. Jika ada pemabuk akan melangkah ke kanan (arah 2). Jika tidak ada berpindah ke loop bersarang setelahnya, algoritma akan memeriksa apakah ada senjata di daerah utara(atas)

pemabuk, jika ada maka pemabuk akan melangkah ke atas(mengembalikan integer 1). Jika tidak ada akan berpindah ke loop bersarang yang terakhir. Algoritma akan memeriksa apakah ada senjata di daerah selatan (bawah) pemabuk, jika ada maka pemabuk akan melangkah ke bawah (fungsi mengembalikan integer 3). Jika tetap tidak ada senjata setelah pemeriksaan sekitar pemabuk, maka fungsi akan mengembalikan integer 0 yang berarti tidak ada senjata di sekitar pemabuk.

Untuk pembagian daerah yang lebih jelas silakan melihat gambar berikut:



Gambar 2. Gambaran pembagian region fungsi cariSenjata dan cariItem

Grid yang diberi titik berwarna biru adalah daerah yang diperiksa pada loop pertama (mengembalikan 4 jika ada senjata). Grid yang diberi titik berwarna hijau adalah daerah yang diperiksa pada loop yang kedua (mengembalikan 2 jika ada senjata). Grid berwarna merah muda adalah daerah yang diperiksa pada loop yang ketiga yang berupa loop bersarang dan akan mengembalikan 1 jika ada senjata di daerah tersebut. Grid yang berwarna merah adalah daerah yang diperiksa pada loop yang terakhir dan akan mengembalikan 3 jika ada senjata di daerah tersebut. Untuk contoh gambar diatas, function cariSenjata akan menghasilkan integer 1, karena senjata ada di daerah yang diberi titik pink. Algoritma ini mempunyai kekurangan yaitu urutan pemeriksaan lebih diutamakan ke arah kiri, kanan, lalu kemudian baru daerah atas, dan bawah. Sehingga jika ada senjata yang terletak 1 kotak di atas pemabuk, namun ada lagi senjata 3 kotak di kiri pemabuk. Maka pemabuk akan bergerak ke kiri bukannya ke atas. Hal ini karena urutan pemeriksaan adalah kiri, kanan, atas, kemudian bawah.

Fungsi selanjutnya adalah fungsi cariItem. Fungsi cariItem sangat mirip *pseudo-code* dan cara kerjanya dengan fungsi cariSenjata sehingga *pseudo-code* dan cara kerjanya tidak perlu dituliskan ulang. Perbedaan di *pseudo-code* hanyalah pada pengecekan grid. Jika di fungsi cariSenjata, di dalam tiap if diperiksa apakah grid kota pedang atau pedangSuper, maka di fungsi cariItem, di dalam tiap if –nya yang diperiksa apakah grid kota tersebut *potion* atau *flyWing.*

Fungsi yang ketiga adalah fungsi cariMusuh. Berbeda algoritmanya dengan cariSenjata atau cariItem, cariMusuh cakupan pemeriksaannya adalah satu kota. Hal ini dikarenakan dia harus terus bergerak walaupun letak musuhnya sangat jauh. Artinya fungsi cariMusuh dan cariPeri(akan dijelaskan kemudian) lebih diutamakan daripada cariSenjata dan cariItem, karena salah satu objektif permainannya adalah membunuh monster atau serigala sebanyak mungkin.

Berikut ini adalah *pseudo-code* fungsi cariMusuh :

```
function cariMusuh(input myKota:Kota,
P:Pemabuk)
{Kota merupakan objek yang memiliki atribut
array dari makhluk-makhluk yang ada di kota
dan matriks of grid dengan tiap grid
menyatakan tembok,jalan,serigala, monster,
senjata, atau item-item lainnya
Keluaran adalah integer 1,2,3,4, atau 0. 1
untuk utara,2 timur,3 selatan,4 barat, }
Deklarasi
  min,dXMin,dYMin:integer
  dX,dY : integer
Algoritma
  min=1
  dXMin=0
  dYMin=0
  dX=0
  dY=0
  for i=2 to myKota.nbMakhluk do
     dXMin=abs(myKota.makhluk[min].x-P.x)
     dYMin=abs(myKota.makhluk[min].y-P.y)
     dX=abs(myKota.makhluk[i].x-P.x)
     dY=abs(myKota.makhluk[i].y-P.y)
     if
(dX<dXMin)and(dY<dYMin)and(myKota.grid[myKo
ta.makhluk[i].x,myKota.makhluk[i].y]=harima
u or monster)then
        min=i;
     endif
  endfor
  if dXMin>dYMin then
     if myKota.makhluk[min].x-P.x>0 then
        →2
     else
        →4
     endif
  else
     if myKota.makhluk[min].y-P.y>0 then
        →3
     else
        →1
     endif
  endif
```

Algoritma cariMusuh diatas idenya adalah mencari jarak terdekat dari makhluk ke –i untuk i=1 to nbMakhluk. Yang dilakukan dalam algoritma ini pertama adalah menghitung jarak untuk tiap makhluk ke pemabuk. Setelah itu dibandingkan jaraknya dengan jarak minimum yang sebelumnya. Jika jarak makhluk yang sedang dihitung lebih kecil dari jarak minimum yang sebelumnya

maka makhluk yang sedang dihitung yang menjadi patokan minimum. Mengapa indeks i mulai dari 1 padahal array untuk menyimpan makhluk dimulai dari 0, hal ini dikarenakan makhluk yang ke 0 adalah peri yang jumlahnya dalam permainan hanya 1. Algoritma cariPeri hanya membandingkan posisi peri dengan pemabuk jika lebih dekat ke kiri maka pemabuk akan berjalan ke kiri, jika lebih dekat ke kanan maka pemabuk akan berjalan ke kanan, jika lebih dekat ke atas maka pemabuk akan berjalan ke atas, sedangkan jika lebih dekat ke bawah maka pemabuk akan berjalan ke atas. Namun ada kekurangan dalam algoritma ini, yaitu jika ada pembatas tembok antara pemabuk dengan makhluk terdekat, yang membuat pemabuk akan berhenti sementara untuk beberapa saat, sampai ada monster atau serigala yang mendekat kepadanya. Namun, hal ini tidak akan berpengaruh banyak terhadap keasyikan bermain karena monster dan serigala yang cukup banyak di dalam kota yang sempit sehingga kemungkinan pemabuk diam karena menabrak tembok jarang terjadi.

Kompleksitas dari Algoritma di atas keseluruhan adalah O(*n*) dengan n maksimum 49 atau banyak makhluk maksimum(dalam program permainan ini 20). Hasil ini didapat dari penjumlahan untuk tiap prosedur atau fungsi. Prosedur pemabukBerjalan kompleksitasnya adalah O(n). Sedangkan kompleksitas waktu asimtotik untuk fungsi cariSenjata dan cariItem adalah O(*n*) karena ada 2 loop bersarang yang jumlah perbandingan isi di dalam loop *fix* 49 kali. Kompleksitas fungsi cariMusuh adalah O(n) untuk n adalah jumlah makhluk dengan jumlah makhluk maksimum di dalam permainan ini dibatasi 20. Kompleksitas fungsi cariPeri adalah O(1). Sehingga jika dijumlahkan kompelksitas waktu asimtotik algoritma *brute force* ini adalah O(n).

3. KESIMPULAN

Algoritma *brute force* lumayan cocok digunakan untuk kecerdasan buatan permainan sederhana seperti dalam permainan *The Legend of DrunkenMan* ini. Hal ini dikarenakan jenis permainan seperti ini yang sederhana. Algoritma *brute force* seperti ini malah lebih mangkus diimplementasikan dalam permainan sederhana karena lebih mudah dimodelkan dan struktur data yang diperlukan lebih kecil, tidak seperti algoritma lain yang lebih canggih yang biasanya memerlukan *tree* dan antrian atau tumpukan. Untuk kompleksitas waktunya sendiri algoritma ini sudah cukup baik karena kompleksitas waktu asimtotiknya yang linier. Hal disebabkan karena memang permainannya yang sederhana dan berukuran kecil. Sehingga sangat disayangkan untuk menggunakan *resource resource* yang banyak untuk merancang kecerdasan buatan yang kompleks untuk permainan ini.

Meskipun begitu tidak ada salahnya berusaha mengembangkan algoritma kecerdasan buatan yang lebih canggih, karena masih adanya kekurangan dalam algoritma *brute force* yang diimplementasikan di permainan ini, yaitu masalah jika pemabuk menabrak tembok yang membuat pemabuk akan terdiam beberapa saat.

REFERENSI

[1] Munir, Rinaldi, “Diktat Kuliah IF2251 Strategi Algoritmik”, Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2007