Penerapan Algoritma Greedy pada Permainan Tower Defense

Tasya - 13515064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515064@std.stei.itb.ac.id

Abstrak—Algoritma *greedy* merupakan salah satu algoritma yang popular untuk memecahkan persoalan optimasi. *Greedy* yang berarti tamak adalah algoritma yang akan mengambil sebuah keputusan terbaik pada suatu *state* tanpa bisa kembali ke *state* sebelumnya apabila ternyata keputusan yang sebelumnya sudah diambil bukan merupakan keputusan optimum global. Oleh karena itu, hasil dari algoritma *greedy* merupakan keputusan optimum lokal dengan harapan keputusan tersebut akan menghasilkan keputusan yang optimum global.

Permainan *Tower Defense* adalah sebuah permainan klasik yang peraturannya cukup mudah, yaitu membunuh monster yang ada sebanyak mungkin sebelum akhirnya monster yang lolos mencapai x buah. Untuk membunuh monster, kita diharuskan membangun sebuah tower yang dapat menembaknya.

Makalah ini akan membahas penggunaan algoritma *greedy* dalam menentukan peletakkan *tower*. Batasan yang ada adalah jumlah uang / koin yang dimiliki oleh pengguna.

Kata kunci: *greedy*, *tower defense*, optimasi.

I. PENDAHULUAN

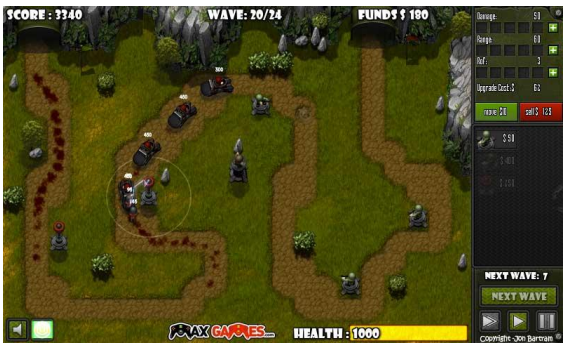
Permainan *Tower Defense* adalah sebuah permainan strategi di mana pemain harus berupaya untuk membunuh monster-monster atau musuh yang akan berjalan pada sebuah *track* tertentu sebanyak mungkin. *Track* tempat jalannya musuh disebut *wave path*. Untuk dapat membunuh monster-monster atau musuh, pemain diharuskan untuk membangun *tower-tower* dan membelinya menggunakan mata uang atau *coin* yang ada dalam permainan.

Secara singkat, objektif dari permainan ini adalah pemain harus mengumpulkan *point* sebanyak mungkin dengan membunuh semua monster atau musuh yang akan muncul. Pemain akan kalah apabila erdapat x buah monster atau musuh yang berhasil lolos mencapai titik *finish* pada *wave path*. Banyaknya monster yang diperbolehkan lolos tergantung pada setiap pembuat permainan *tower defenese* ini.

Ketika permainan dimulai, pemain akan dibawa ke sebuah lahan kosong yang terdapat sebuah jalan (yang kemudian akan berperan sebagai *wave path*). Selain pada *wave path* tersebut, pemain dapat membangun *tower-tower* untuk membunuh monster yang akan muncul. Dalam beberapa waktu saat permainan dimulai, monster tidak akan muncul karena sistem member pemain waktu untuk membangun sejumlah *tower* untuk bertahan terlebih dahulu.

Pada umumnya, ada sebuah batas maksimal jumlah *tower* yang dapat dibangun oleh pengguna agar permainan tetap menjadi seru. Selain itu, pemain juga diberi sejumlah mata uang atau *coin* sebagai modal awal, kemudian pemain akan mendapat *coin* tambahan apabila berhasil membunuh monster-monster yang ada. Guna dari *coin* tersebut adalah sebagai mata uang untuk membangun dan membeli *tower*.

Agar permainan lebih menantang, semakin lama monster akan muncul semakin cepat (waktu *spawn* monster akan semakin kecil) sehingga pada suatu waktu, *wave path* akan penuh dengan monster. Sebelum *wave path* mulai penuh, diharapkan pemain sudah mempunyai sejumlah *tower* yang cukup dengan lokasi *tower* yang strategis agar dapat mengatasi banyaknya jumlah monster yang akan muncul. Berikut adalah contoh salah satu antarmuka dari permainan *Tower Defense*:



Gambar 1 Contoh Antarmuka Salah Satu Permainan *Tower Defense*

Pada permainan ini, diperlukan strategi untuk menentukan peletakan setiap *tower* yang efektif dalam batasan jumlah *coin* yang dimiliki maupun jumlah maksimal *tower* yang bisa dibangnun. Pada makalah ini, strategi penempatan *tower* akan dilakukan berdasarkan algoritma ***greedy***. Algoritma *greedy* berusaha menghasilkan solusi optimal, namun tidak selalu dapat menghasilkan solusi yang optimal karena prinsip algoritma ini adalah ***"take what you can get now".***

II. DASAR TEORI

Algoritma *greedy* akan digunakan sebagai dasar penempatan *tower* untuk mencapai hasil yang maksimal dengan berbagai keterbatasan yang ada, yaitu jumlah maksimal *tower* yang dapat dibangun dan keterbatasan jumlah *coin* yang dimiliki oleh pemain.

Persoalan optimasi pada permainan ini termasuk ke dalam persoalan maksimalisasi, yaitu dapat membunuh monster pada daerah sebanyak-banyaknya dengan batasan jumlah maksimal *tower* yang dapat dibuat dan batasan *coin* untuk membeli *tower*.

Berdasarkan arti harafiahnya, *greedy* berarti tamak, rakus, atau loba. Sesuai dengan artinya, prinsip dari algoritma *greedy* adalah memilih keputusan terbaik pada setiap *state* atau langkah (akan menghasilkan keputusan optimum lokal) dengan harapan akan menghasilkan keputusan optimum global. Namun, apabila keputusan terbaik pada suatu langkah ternyata tidak mengarahkan pada optimum global, tidak dapat dilakukan peninjauan kembali (*backtracking*) terhadap keputusan tersebut. Oleh karena itu, hasil keputusan dari algoritma *greedy* mungkin tidak optimal.

Algoritma *greedy* memiliki beberapa elemen yang harus dupenuhi, yaitu sebagai berikut:

- Himpunan Kandidat
  Himpunan kandidat adalah semua ruang sampel yang dapat dipilih sebagai solusi.
- Himpunan Solusi
  Himpunan solusi adalah himpunan bagian dari himpunan kandidat yang merupakan solusi dari permasalahan yang ada.
- Fungsi Seleksi
  Fungsi seleksi merupakan unsure khas dari algoritma ini. Fungsi ini digunakan untuk menyeleksi anggota-anggota dari himpunan kandidat yang akan termasuk ke dalam tahap selanjutnya yang akan diperiksa menggunakan fungsi kelayakan agar dapat masuk ke dalam himpunan solusi. Misalnya, mengambil tempat dengan bidikan paling banyak, mengambil koin bernilai paling tinggi, dsb.
- Fungsi Layak
  Fungsi layak digunakan untuk memeriksa apakah solusi yang dipilih merupakan solusi yang layak untuk dimasukkan ke dalam himpunan solusi atau tidak. Misalnya memeriksa apakah *coin* yang dimiliki oleh pemain masih cukup untuk membangun *tower* baru.
- Fungsi Objektif
  Fungsi objektif adalah solusi optimum yang dihasilkan. Misalnya semua *wave path* sudah ditangani oleh seluruh *tower* yang ada, jumlah koin yang digunakan minimum, dsb.

Berikut adalah *pseudocode* dari algoritma *greedy*:

```
function greedy(input C: himpunan_kandidat)→ himpunan_kandidat
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy
  Masukan: himpunan kandidat C
  Keluaran: himpunan solusi yang bertipe himpunan_kandidat
}
Deklarasi
    x : kandidat
    S : himpunan_kandidat

Algoritma:
    S ← {}   { inisialisasi S dengan kosong }
    while (not SOLUSI(S)) and (C ≠ {} ) do
       x ← SELEKSI(C)        { pilih sebuah kandidat dari C}
       C ← C - {x}               { elemen himpunan kandidat berkurang satu }
       if LAYAK(S ∪ {x}) then
           S ← S ∪ {x}
       endif
    endwhile
    {SOLUSI(S) or C = {} }

    if SOLUSI(S) then
       return S
    else
       write('tidak ada solusi')
    endif
```

Walaupun algoritma *greedy* jauh lebih efektif dibandingkan algoritma *brute force* dalam hal optimasi waktu, algoritma ini memiliki kekurangan. Kekurangannya adalah algoritma *greedy* tidak selalu menghasilkan solusi optimum global. Hal ini disebabkan oleh algoritma *greedy* tidak memeriksa semua kemungkinan solusi yang mungkin dan hanya mnegmabil solusi terbaik relative (yang dapat disebut sebagai optimum lokal) pada setiap langkah atau *stage*-nya.

Namun karena algoritma *greedy* ini hampir selalu menghasilkan solusi yang mendekati optimal global, maka algoritma ini sangat cocok digunakan untuk

memecahkan persoalan optimasi. Agar kemungkinan hasil optimum global didapatkan, hal yang dapat dilakukan adalah memilih fungsi seleksi yang baik karena fungsi seleksilah yang menentukan solusi mana yang akan diambil dan dimasukkan ke dalam himpunan solusi pada setiap langkah atau *stage*.

Hasil dari algoritma *greedy* mungkin lebih dari satu sehingga solusi yang dihasilkan mungkin berbeda-beda. Karena algoritma *greedy* tidak menjamin optimalitas solusi, maka pemilihan fungsi seleksi pada algoritma ini menjadi sangat penting.

III. IMPLEMENTASI

Pada kasus ini, terdapat beberapa asumsi agar analisis yang dilakukan pada permainan ini lebih jelas, yaitu:

1. Semua *tower* yang digunakan sama, tidak dapat ditingkatkan kemampuannya dan hanya terdapat satu jenis *tower*.
2. Radius serangan tiap *tower* adalah 1 blok *Area of Effect* (AOE).
3. *Tower* yang sudah dibangun tidak dapat dipindahkan.
4. Hanya dapat membangun satu *tower* pada satu blok.
5. Monster atau musuh berjalan pada suatu jalur yang tetap (*wave path*).
6. Monster tidak dapat bergerak mundur.
7. Kecepatan jalan monster statis atau tidak dapat berubah.
8. Monster tidak dapat menyerang *tower*.
9. Jalur yang dilalui monster hanya satu arah tanpa ada percabangan.
10. Pemain dinyatakan menang apabila pada akhir waktu yang ditentukan, monster yang tidak terbunuh tidak mencapai x monster.
11. Area permainan dibuat dengan matriks dengan ukuran 10 blok x 10 blok.
12. Jumlah *tower* yang dapat dibangun tergantung pada jumlah *coin* yang dimiliki pemain.

Elemen-elemen algoritma *greedy* pada permasalahan optimasi peletakan *tower* pada permainan *Tower Defense* adalah sebagai berikut:

- Himpunan Kandidat: semua blok yang selain *wave path*.
- Himpunan Solusi: himpunan blok yang dipilih sebagai tempat meletakkan *tower*, jika memungkinkan semua *wave path* ditangani oleh minimal satu *tower*.
- Fungsi Seleksi: blok yang dipilih untuk meletakkan *tower* yang memiliki AOE paling banyak.
- Fungsi Layak: blok yang dipilih bukan merupakan *wave path*, *coin* yang dimiliki oleh pemain masih mencukupi, dan pada titik tersebut belum dibangun *tower* lain.
- Fungsi Objektif: memaksimalkan skor yang didapat dengan memperoleh (AOE) sebesar mungkin untuk *tower* dapat menyerang monster atau musuh.

Sebelum *tower* dibangun pada titik yang dipilih berdasarkan hasil dari algoritma *greedy*, akan diperiksa apakah *coin* yang dimiliki oleh pemain masih mencukupi untuk membangun sebuah *tower*, lalu blok yang dipilih akan dicek apakah sudah terdapat *tower* lain atau belum.

Berikut adalah contoh ilustrasi sebuah area permainan *Tower Defense*:

0	1	2	3	4	5	6	7	8	9	10
A										
B										
C										
D										
E										
F										
G										
H										
I										
J				START					END	

Gambar 2 Ilustrasi Area Permainan *Tower Defense*

Keterangan:

= *wave path*

= titik yang boleh dibangun *tower*

Langkah-langkah untuk mendapatkan Himpunan Solusi dari blok-blok Himpunan Kandidat adalah sebagai berikut:

1. Mengambil blok-blok pada area permainan yang memenuhi fungsi kelayakan (dalam hal ini dapat dibangun *tower*) dengan fungsi *generateArea* dan memberikan nilai representasi tiap titik.

0	1	2	3	4	5	6	7	8	9	10
A	0	0	0	0	0	0	0	0	0	0
B	9	9	9	9	9	9	0	0	0	0
C	9	0	0	0	0	9	9	0	0	0
D	9	0	0	0	0	0	9	9	0	0
E	9	0	9	9	9	0	0	9	9	0
F	9	0	9	0	9	0	0	0	9	0
G	9	9	9	0	9	0	0	0	9	0
H	0	0	0	0	9	0	0	9	9	0
I	0	0	9	9	9	0	0	9	0	0
J	0	0	9	0	0	0	0	9	0	0

Gambar 3 Area Permainan yang Sudah Diberi Nilai Representasi

Keterangan:
0: merepresentasikan wilayah yang dapat dibangun *tower*
9:merepresentasikan *wave path*, tidak dapat dibangun *tower*.
1: merepresentasikan blok yang sudah dibangun *tower*.
2: merepresentasikan wilayah *wave path* yang sudah ditangani oleh *tower*.

2. Himpunan blok yang layak bangun diambil dengan fungsi *makeEnable*.
   Didapatkan himpunan T, yaitu himpunan yang berisi blok-blok yang dapat didirikan *tower*.

   T={{1,A}, {1,H}, {1,I}, {1,J}, {2,A}, {2,C}, {2,D}, {2,E}, {2,F}, {2,H}, {2,I}, {2,J}, {3,A}, {3,C}, {3,D}, {3,H}, {4,A}, {4,C}, {4,D}, {4,F}, {4,G}, {4,H}, {4,J}, {5,A}, {5,C}, {5,D}, {5,J}, {6,A}, {6,D}, {6,E}, {6,F}, {6,G}, {6,H}, {6,I}, {6,J}, {7,A}, {7,B}, {7,E}, {7,F}, {7,G}, {7,H}, {7,I}, {7,J}, {8,A}, {8,B}, {8,C}, {8,F}, {8,G}, {9,A}, {9,B}, {9,C}, {9,D}, {9,I}, {9,J}, {10,A}, {10,B}, {10,C}, {10,D}, {10,E}, {10,F}, {10,G}, {10,H}, {10,I}, {10,J}}

3. Pada himpunan kandidat T dapat dicari AOE dari blok tersebut yang paling banyak dengan fungsi *findMaxAoe*, *return* dari fungsi ini adalah blok yang memiliki AOE paling besar sesuai dengan algoritma *greedy*. Apabila ada dua blok yang memiliki AOE yang sama, maka akan didapatkan blok yang ditemukan terlebih dahulu.
   Proses pencarian blok dengan AOE terbesar adalah dengan mencari blok yang di sekitarnya memiliki blok dengan nilai '9' paling banyak.
   Pada contoh area permainan di atas, didapatkan blok {2,F} dengan AOE terbesar yaitu 7 blok.

4. *Tower* dibangun pada titik yang telah ditemukan pada langkah 3 dengan prosedur *buildTower*.
   Pada algoritma, proses pembangunan *tower* dapat dilakukan dengan mengubah representasi nilai pada blok yang terpilih dari '0' menjadi '1'. Selain itu, daerah *wave path* yang sudah ditangani oleh *tower* yang baru saja dibuat akan diubah representasinya dari '9' menjadi '2'.

0	1	2	3	4	5	6	7	8	9	10
A	0	0	0	0	0	0	0	0	0	0
B	9	9	9	9	9	9	0	0	0	0
C	9	0	0	0	0	9	9	0	0	0
D	9	0	0	0	0	0	9	9	0	0
E	2	0	2	9	9	0	0	9	9	0
F	2	1	2	0	9	0	0	0	9	0
G	2	2	2	0	9	0	0	0	9	0
H	0	0	0	0	9	0	0	9	9	0
I	0	0	9	9	9	0	0	9	0	0
J	0	0	9	0	0	0	0	9	0	0

Gambar 4 Representasi Area Permainan Setelah Langkah 3 dan 4

Keterangan:
= AOE dari blok *tower* dengan representasi '2'
= Lokasi *tower* dengan representasi '1'

5. Langkah satu sampai dengan empat dialkukan terus menerus hingga *coin* dari pemain habis.

PSEUDOCODE

1. Fungsi generateArea

```
function generateArea(input
P:battleground,point: titik) → Integer

{fungsi ini akan mengembalikan nilai
integer dengan representasi 1 untuk
tower yang sudah dibangun, 2 untuk
AOE tower, 9 untuk jalur musuh yang
melintas, dan 0 untuk wilayah layak
bangun tower pada variabel nilai,
lanjut di halaman berikut}
```

```
if (titik adalah wavepath yang belum
ditangani tower manapun) then
    → 9
else if (titik adalah wavepath yang
sudah ditangani satu tower atau
lebih) then
    → 2
else if (titik sudah ditempati
sebuah tower) then
    → 1
else {titik bukan wavepath dan belum
ada tower yang dibangun}
    → 0
endif
```

2. Fungsi makeEnable

```
function makeEnable(input
P:battleground) → array of point

{fungsi ini mengembalikan titik di
battleground yang layak dibangun
tower atau memiliki nilai 0 pada
representasi titik menggunakan fungsi
generateArea dan variabel nilai}

T: array of point

T ← {} {inisialisasi array kosong}
i iterate [A..J]
    j iterate [1..10]
        if (titik = '0') then
            T ← T + titik
        endif

→ T
```

3. Fungsi findMaxAce

```
function findMaxAoe(input T : array
of point, P:battleground) → point

{fungsi ini untuk mengembalikan titik
dengan radius AOE/radius erang
terbesar dan mengeluarkan titik
tersebut dari list atau himpunan
titik layak bangun}

i: Integer
retval: point
retval ← T[0] {inisialisasi}

i iterate [1..T.size]
    if (T[i].AOE > retval) then
        retval ← T[i]
    endif

→ retval
```

4. Fungsi buildTower

```
procedure buildTower(input
P:battleground, place:point)
{titik place dibangun tower dan
variabel nilai diberi nilai '1'.
Radius 1 blok dari titik place
diberi nilai '2' untuk menghandle
terjadinya kasus penempatan tidak
optimal}

battleground : P
list : array of point
nilai : array[1..10,1..10] of
integer
gold : integer
towerPrice : integer

list <- makeEnable(P)
while (not isEmpty(list) and
gold>towerPrice)

buildTower(P,findMaxAoe(list,P))
```

IV. KESIMPULAN

Dalam persoalan optimasi, algoritma *greedy* merupakan salah satu algoritma yang sering digunakan. Walaupun ada kemungkinan solusi yang dihasilkan oleh algoritma *greedy* bukan merupakan solusi yang paling optimal, namun biasanya solusi tersebut sudah mendekati atau menghampiri hasil yang paling optimal. Selain itu, salah satu kelebihan algirutma *greedy* lainnya adalah implementasinya yang sederhana.

Namun demikian, jika masalah yang dihadapi mengharuskan didapatkannya solusi yang paling optimum, sangat beresiko untuk menggunakan algoritma *greedy* karena harus dapat dibuktikan secara matematis terlebih dahulu bahwa hasil yang didapatkan dari algoritma tersebut merupakan solusi yang paling optimum. Jika tidak dapat dibuktikan secara matematis, maka tidak dapat dijamin bahwa algoritma *greedy* yang telah dibuat akan menghasilkan solusi yang optimum untuk semua kasus atau masalah yang mungkin ditemui.

Untuk mengatasi persoalan algoritma *greedy* yang terkadang tidak menghasilkan solusi optimum global, dapat digunakan algoritma lain yaitu algoritma *exhaustive search*. Dengan menggunakan algoritma *exhaustive search*, semua solusi yang mungkin dicapai akan didata satu-satu, lalu kemudian diseleksi sehingga ditemukan satu solusi yang paling optimal. Dengan cara ini, solusi optimal pasti tercapai karena semua solusi yang mungkin dicatat.

Namun untuk kasis optimasi yang tidak diharuskan untuk mendapatkan solusi yang benar-benar paling optimum, algoritma *greedy* sangat cocok untuk diterapkan. Algoritma *greedy* dapat menjadi pilihan terbaik dibandingan algoritma *brute force* yang kompleksitasnya jauh lebih besar. Tentu saja algoritma *greedy* akan lebih cepat untuk dieksekusi.

REFERENSI

[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung. 2009.

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 Mei 2017

Tasya / 13515064