Penggunaan Algoritma Greedy untuk Mencari Solusi Optimal dalam Permainan Brick Breaker

Nanda Ekaputra Panjiarga – 13509031
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509031@std.stei.itb.ac.id

ABSTRAK

***Brick Breaker* adalah suatu permainan dimana pemainnya harus mengeliminasi kumpulan kotak-kotak sebanyak mungkin. Kumpulan kotak yang harus dieliminasi harus memiliki jumlah anggota di atas angka tertentu. Makalah ini akan membahas tentang pengaplikasian algoritma *Greedy* untuk mencari solusi paling optimal dalam permainan *Brick Breaker* ini, serta menganalisa optimalitas dari solusi yang dihasilkan.**

Index Terms*—*Brick Breaker*, Algoritma *Greedy

I. PENDAHULUAN

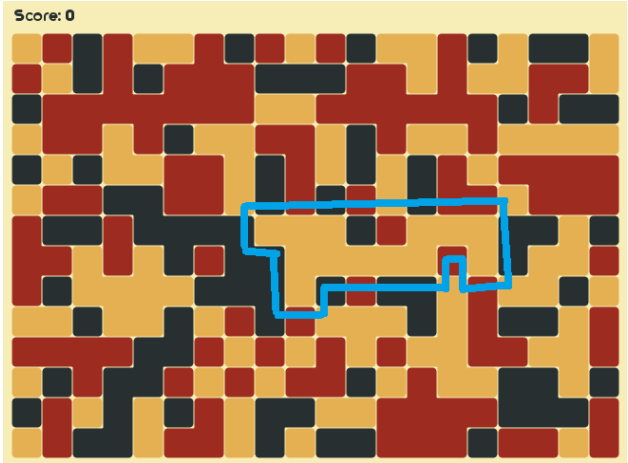
1.1. Pengenalan Brick Breaker

Brick Breaker adalah salah satu permainan yang bertipe puzzle. Konsep dari permainan ini sangatlah sederhana dan sudah sering diadaptasi ke berbagai macam nama game, diantaranya *5Connect*, *Cube Buster*, *Crashdown*, dan lain-lain. Permainan-permainan ini dapat kita temui di situs-situs penyedia permainan gratis seperti *Armorgames*, *Miniclip*, *HTML5games.net*, dan masih banyak lagi. Biasanya permainan ini disajikan dalam format *flash game*, *Java applet*, atau *HTML5*.

1.1.1. Konsep dasar Brick Breaker

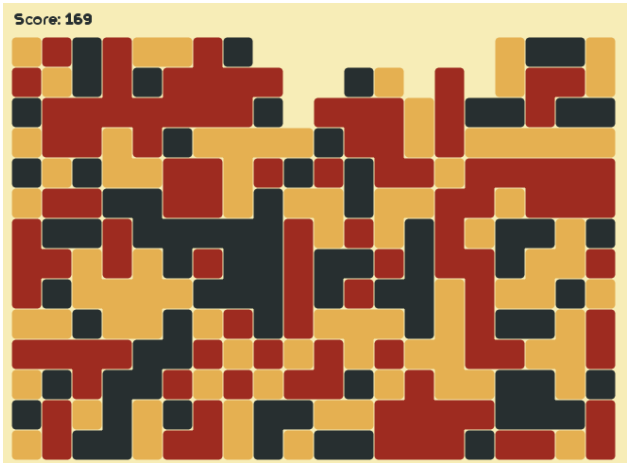
Konsep dasar dari permainan ini sangat sederhana. Dalam Brick Breaker, terdapat sebuah tumpukan kotak-kotak dengan warna yang berbeda-beda. Masing-masing kotak berukuran sama, yakni sebuah persegi. Kotak-kotak tersebut tersusun dalam sebuah tumpukkan yang merepresentasikan matriks dengan ukuran tertentu.

Tiap kotak tersebut memiliki 'kotak tetangga', yaitu kotak yang tepat bersentuhan langsung di samping kanan, samping kiri, atas, atau bawah. Apabila sebuah kotak memiliki kotak tetangga yang berwarna sama dengannya, maka mereka akan dianggap sebagai satu kesatuan.



Gambar 1: *Screenshot* permainan saat mulai

Contohnya dapat dilihat dalam *screenshot* permainan 5*Connect* di atas. Pada bagian yang ditandai dengan garis biru, kotak-kotak kuning tersebut tersusun saling bertetangga satu sama lain sehingga dapat dianggap sebagai sebuah kesatuan.



Gambar 2: *Screenshot* permainan setelah satu langkah

Pada tiap langkahnya pemain diharuskan mencari susunan kotak seperti ini kemudian menyeleksi bagian tersebut. Susunan kotak yang terdiri dari n atau lebih kotak akan bisa diseleksi. Nilai n ini bergantung kepada aturan permainan yang ditentukan penciptanya, biasanya bernilai 3 atau lebih. Ketika diseleksi, maka bagian tersebut akan hilang (gambar 2). Hal ini akan

mengakibatkan susunan kotak-kotak diatasnya akan jatuh dan mengisi ruang kosong yang ditinggalkan oleh susunan kotak yang telah diseleksi.

Dengan demikian, pada setiap langkahnya akan dihasilkan kondisi yang benar-benar baru. Muncul pola-pola baru yang terbentuk dari hasil penghilangan beberapa kotak yang terseleksi.

Kegiatan mencari pola dan menyeleksinya ini akan terus dilakukan oleh pemain hingga mencapai tujuan tertentu. Tujuan atau *goal* akhir dari permainan ini bervariasi, tergantung dari judul permainan yang dimainkan.

1.1.2. Variasi Brick Breaker

Konsep dasar Brick Breaker telah digunakan dan dikembangkan ke berbagai macam judul permainan. Yang membedakan masing-masing dari judul tersebut, selain tampilannya, adalah tujuan akhir dari permainan.

Berikut akan dibahas dua contoh judul permainan dengan konsep Brick Breaker yang memiliki tujuan akhir berbeda.

1.1.2.1. *5Connect*

Tujuan akhir dari permainan *5Connect* adalah menghabiskan seluruh kombinasi kotak dengan anggota berjumlah di atas n buah. Pada *5Connect*, n bernilai 2.

Gambar 3: Akhir dari permainan *5Connect*

Setelah tidak dapat dihasilkan lagi kombinasi kotak tersebut, maka permainan akan berakhir. Pemain akan mendapatkan nilai yang semakin optimal apabila kotak-kotak yang tersisa semakin sedikit (gambar 3).

1.1.2.2. *CrashDown*

Konsep permainan *CrashDown* memiliki keseragaman dengan game *Tetris*. Pemain harus mempertahankan tumpukan blok yang ada agar tidak melampaui batas yang telah ditentukan.

Pada selang waktu tertentu, akan muncul tambahan blok dengan pola acak di bagian bawah tumpukan blok. Pemain harus menghilangkan tumpukan blok yang ada dengan cepat agar tumpukan tersebut tidak melampaui batas paling atas layar.

Gambar 4: Permainan berakhir dengan baik

Gambar 5: Permainan gagal

Permainan berakhir bila batas atas telah dilampaui (gambar 5) atau waktu permainan telah selesai (gambar 4).

1.2. Algoritma Greedy

1.2.1. Definisi Algoritma Greedy

Algoritma Greedy merupakan salah satu metode yang paling populer untuk menyelesaikan persoalan optimasi. Yang dimaksud dengan persoalan optimasi adalah persoalan yang mencari solusi optimal, baik yang bersifat maksimal (maksimalisasi), atau minimal (minimalisasi).

Dalam bahasa inggris, Greedy berarti rakus atau tamak. Hal ini mencerminkan prinsip dari algoritma ini, yaitu "*take what you can get now!*". Algoritma ini akan membentuk solusi langkah per langkah. Pada setiap langkah, algoritma ini akan mengeksplorasi segala kemungkinan pilihan yang ada. Dari seluruh kemungkinan pilihan tersebut, akan diambil pilihan yang paling baik untuk setiap langkahnya. Pilihan terbaik ini disebut solusi optimal lokal.

Harapannya, dengan terus menerus mengambil pilihan optimal terbaik untuk tiap langkahnya, akan dihasilkan solusi global (solusi dari keseluruhan langkah) yang juga optimal.

Kekurangan dari algoritma ini adalah tidak adanya perhatian terhadap konsekuensi dari tiap langkah yang diambil, sehingga bisa saja rangkaian langkah yang dihasilkan bukan merupakan solusi optimal global.

1.2.2. Penggunaan Algoritma Greedy

Dalam permainan Brick Breaker ini, tiap langkah akan menghasilkan kondisi yang benar-benar baru. Dengan demikian, tiap langkah akan memiliki solusi optimalnya masing-masing.

Dengan algoritma Greedy, pendekatan untuk mencari solusi optimal global akan dilakukan dengan cara mencari solusi optimal lokal untuk tiap langkahnya.

Definisi dari kondisi optimal yang dimaksud akan bervariasi, namun semuanya berdasarkan dari salah satu, atau kombinasi dari kedua variasi tujuan utama permainan dengan konsep Brick Breaker.

II. LANDASAN TEORI

2.1. Teori Dasar Algoritma Greedy

Dalam Algoritma Greedy terdapat elemen-elemen berikut:

1. Himpunan kandidat, C
2. Himpunan solusi, S
3. Fungsi seleksi, SELEKSI()
4. Fungsi kelayakan, LAYAK()
5. Fungsi obyektif, SOLUSI()

Himpunan kandidat merupakan himpunan kandidat solusi yang merepresentasikan opsi yang ditawarkan setiap langkahnya. Himpunan solusi merepresentasikan langkah apa saja yang telah diambil hingga akhir algoritma.

Fungsi seleksi menyeleksi kumpulan pilihan yang ada dan mengembalikan pilihan yang akan menghasilkan solusi optimal. Fungsi kelayakan bertindak sebagai pembatas, apakah pilihan yang dipilih layak sesuai ketentuan dari persoalan. Kombinasi dari kedua fungsi ini akan menghasilkan solusi optimum untuk setiap langkahnya. Solusi-solusi optimum lokal tersebut akan menjadi hasil dari sebuah fungsi objektif, yaitu persoalan utama yang ingin diselesaikan.

Berikut adalah skema umum dari algoritma Greedy:

```
function greedy(input C: himp_kandidat)
→ himp_kandidat
{   Mengembalikan   solusi   dari   persoalan
optimasi dengan algoritma greedy.
Masukan: himpunan kandidat C
Keluaran:   himp.   Solusi   yang   bertipe
himp_kandidat }

Deklarasi
x : kandidat
S : himp_kandidat

Algoritma
S ← { } {inisialisasi S dengan kosong}
while (not SOLUSI(S)) and (C!={}) do
        x ← SELEKSI(C){pilih kandidat dari C}
        C ← C – {x} {elemen C berkurang satu}
        if LAYAK(S U {x}) then
                S ← S U {x}
        endif
endwhile
{SOLUSI(S) atau C kosong}

if SOLUSI(S) then
        return S
else
        {tidak ditemukan solusi global}
endif
```

Pertama-tama, algoritma greedy akan mengiterasi semua elemen himpunan kandidat yang ada dan menyeleksi kandidat yang akan menghasilkan solusi optimal lokal. Kemudian dilakukan pengecekan dahulu terhadap kandidat tersebut dengan fungsi kelayakan, apakah sudah sesuai dengan ketentuan persoalan. Jika sudah cocok, maka kandidat dimasukkan ke dalam himpunan solusi.

Hal ini terus dilakukan periterasi sampai elemen di himpunan kandidat habis, solusi telah ditemukan atau telah mencapai kondisi akhir yang diinginkan.

III. METODE PEMECAHAN MASALAH

Untuk menggunakan algoritma greedy, kita perlu mendefinisikan kondisi apa yang melambangkan situasi optimal. Dalam makalah ini kita akan membahas dua kondisi optimal, yaitu kondisi optimal berdasarkan jumlah kotak yang bertetanggaan, dan kondisi optimal berdasarkan posisi kotak.

3.1. Greedy by Jumlah Kotak

Bagaimanapun bentuk dan variasinya, tujuan umum dari game dengan konsep Brick Breaker adalah menghilangkan sebanyak mungkin kotak dalam permainan.

Karena itulah dapat diasumsikan semakin banyak kotak yang dihilangkan tiap giliran, akan semakin optimum solusi yang dihasilkan.

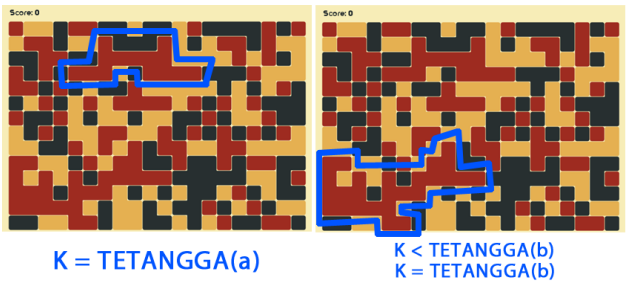
Algoritma greedy by jumlah kotak akan mencari daerah mana yang memiliki jumlah kotak bertetanggaan dan berwarna sama paling banyak pada setiap gilirannya.

Misalkan area permainan direpresentasikan oleh matriks ixj yang masing-masing elemennya memuat informasi warna kotak pada koordinat ixj. Algoritma ini

akan mengiterasi seluruh isi matriks ixj tersebut untuk mencari area ketetanggaan yang paling besar. Untuk bisa diseleksi dan dihilangkan, diasumsikan bahwa area ketetanggaan tersebut harus beranggotakan minimal 3 kotak. Berikut skema dari algoritma Greedy by Jumlah Kotak untuk persoalan Brick Breaker:

```
function LAYAK(input K: himp_solusi) → boolean
{ mengembalikan true jika elemen K > 3}
Deklarasi

Algoritma
return (jumlah elemen K > 3)


function greedybyjumlah(input M: matriks persoalan)
→ himp_solusi
{ Mengembalikan solusi lokal dari persoalan optimasi dengan algoritma greedy.
Masukan: Matriks persoalan brick breaker
Keluaran: himp. Solusi yang berisi elemen matriks, himpunan kosong bila tidak ada solusi optimum lokal
Prekondisi: Matriks tidak kosong}

Deklarasi
S,K : himp_solusi

Algoritma
K ← {}
for (setiap elemen M) do
   S ← TETANGGA(elemen M)
   if (jumlah elemen S >= jumlah elemen K) and LAYAK(S) then
      K ← S
   endif
endfor

if LAYAK(K) then
        return K
else
        return {}{tidak ditemukan area dengan kotak>3}
endif
```

Himpunan kandidat dari persoalan ini adalah elemen-elemen matriks tersebut. Pada setiap elemen, akan diaplikasikan fungsi TETANGGA() yang berperan sebagai fungsi seleksi. Fungsi tersebut akan mencari seluruh tetangga dari elemen yang sedang diperiksa, kemudian mengembalikan himpunan yang berisi elemen-elemen matrix yang saling berketetanggaan.

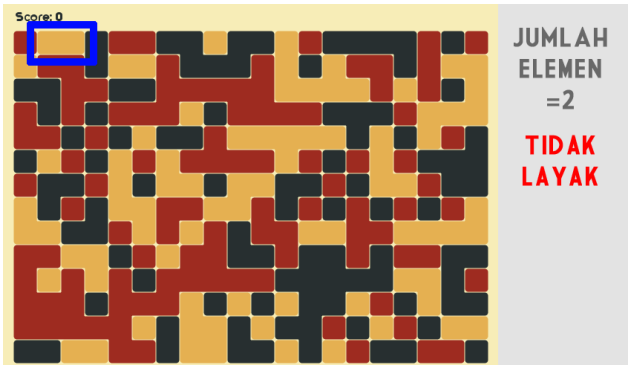


Gambar 6: Penggunaan fungsi TETANGGA() sebagai fungsi seleksi

Untuk implementasi fungsi TETANGGA(), bisa digunakan algoritma BFS, DFS, atau *backtracking* yang diluar dari lingkup pembahasan makalah ini.

Pada setiap iterasi per kotak, akan dicek jumlah elemen dari hasil fungsi TETANGGA(). Algoritma ini akan mencari solusi yang memiliki jumlah elemen terbanyak. Setelah itu solusi tersebut diperiksa menggunakan fungsi LAYAK(). Fungsi ini berperan sebagai fungsi kelayakan. Dalam persoalan ini, solusi dianggap layak apabila terdiri dari tiga atau lebih anggota.

Seperti pada contoh gambar di bawah ini, area dengan dua kotak akan dianggap tidak layak.



Gambar 7: Penggunaan fungsi kelayakan

Fungsi greedybyjumlah diatas akan digunakan setiap kali pemain melakukan turn, dan memerlukan prekondisi bahwa matriks persoalan tidak kosong. Apabila fungsi ini mengembalikan himpunan kosong, tandanya sudah tidak ada lagi area yang dapat diseleksi dan dihilangkan.

3.2. Greedy by Posisi Kotak

Selain jumlah kotak yang akan dihilangkan, posisi dari kotak yang akan dihilangkan seharusnya juga mendapatkan pertimbangan.

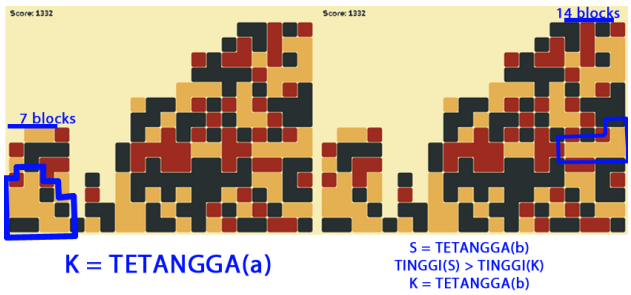
Sebagai contoh, pada permainan *CrashDown*, apabila pemain terus menerus menghilangkan kotak hanya pada kolom tertentu, maka kolom lain yang tidak tersentuh akan tetap meninggi dan akhirnya menggagalkan permainan. Karena itulah dibutuhkan fungsi seleksi yang memperhatikan ketinggian maksimum kolom dari kotak-kotak yang akan dihilangkan.

Asumsikan seluruh peraturan dan ketentuan persoalan sama dengan contoh pada bagian Greedy by Jumlah Kotak. Berikut skema dari algoritma Greedy by Posisi Kotak untuk persoalan Brick Breaker:

```
function greedybyposisi (input M: matriks persoalan)
→ himp_solusi
{ Mengembalikan solusi lokal dari persoalan optimasi dengan algoritma greedy.
Masukan: Matriks persoalan brick breaker
Keluaran: himp. Solusi yang berisi elemen matriks, himpunan kosong bila tidak ada solusi optimum lokal
Prekondisi: Matriks tidak kosong}

Deklarasi
S,K : himp_solusi

Algoritma
K ← {}
for (setiap elemen M) do
   S ← TETANGGA(elemen M)
   if (TINGGI(S) >= TINGGI(K)) and LAYAK(S) then
      K ← S
   endif
endfor

if LAYAK(K) then
        return K
else
        return {}{tidak ditemukan area dengan kotak>3}
endif
```

Sama seperti sebelumnya, himpunan kandidat dari persoalan ini adalah elemen-elemen matriks tersebut. Pada setiap elemen juga diaplikasikan fungsi TETANGGA() yang berperan mencari seluruh tetangga dari elemen yang sedang diperiksa, kemudian mengembalikan himpunan yang berisi elemen-elemen matrix.

Namun, di sini terdapat fungsi seleksi TINGGI() yang akan mengembalikan tinggi maksimal dari kolom tempat elemen-elemen matriks berada. Algoritma ini memastikan bahwa elemen dengan tinggi kolom lebih besar memiliki prioritas yang lebih tinggi untuk dihapus. (Gambar 8)



Gambar 8: Elemen dengan tinggi kolom lebih besar memiliki prioritas yang lebih tinggi untuk dihapus.

IV. ANALISIS

Implementasi konsep algoritma greedy terhadap persoalan Brick Breaker ini memiliki sedikit perbedaan dengan skema algoritma greedy pada umumnya.

Umumnya, algoritma greedy memiliki sebuah set kandidat tetap yang akan berkurang tahap demi tahap. Namun, pada persoalan ini himpunan kandidatnya berubah pada setiap langkah, sehingga diperlukan pendekatan baru (yang masih berupa implementasi algoritma greedy) untuk menyelesaikan persoalan tersebut.

Dengan sedikit modifikasi dari konsep awal, algoritma ini akhirnya melakukan proses perbandingan di tiap langkah, dengan memasukkan kembali kondisi persoalan saat itu (dalam hal ini *field* dari permainan).

Algoritma greedy belum menjamin didapatnya solusi persoalan yang optimum secara global. Algoritma ini hanya dapat memberikan aproksimasi tentang nilai optimal tersebut. Hal ini disebabkan oleh sifat dari setiap langkah yang *contiguous*, atau dengan kata lain saling berkesinambungan.

Pada persoalan ini, dihapusnya beberapa blok kotak akan menghasilkan kondisi baru yang akan langsung dianalisa kembali oleh algoritma greedy. Dalam kenyataannya, permainan ini membutuhkan strategi yang lebih kompleks daripada sekedar mencari kotak terbanyak atau kotak dengan tinggi kolom tertinggi. Permainan ini juga membutuhkan perencanaan guna memastikan tidak ada kotak yang pada akhirnya tidak dapat dihapus.

Strategi yang kompleks tersebut bisa dikembangkan dengan mengombinasikan algoritma Greedy dengan algoritma backtracking, DFS, BFS, atau Dynamic Programming. Dengan memanfaatkan algoritma lainnya, Greedy masih bisa terus memberikan solusi optimal lokal tiap turnnya, sementara algoritma lain bertugas untuk mencari kombinasi solusi yang benar-benar optimal secara global.

V. KESIMPULAN

Meskipun sudah dapat memberikan aproksimasi tentang kondisi paling optimal yang merupakan solusi dari permainan Brick Breaker ini, algoritma Greedy tidak menjamin untuk memberikan solusi yang benar-benar optimal secara global.

Diperlukan kombinasi algoritma lainnya untuk mendapatkan solusi yang mutlak optimal dari permainan Brick Breaker.

REFERENSI

[1] http://wwwc3.lanl.gov/mega-math/gloss/compute/greedy.html 05 Desember 2012

[2] http://www.informatika.org/~rinaldi/ 05 Desember 2012

[3] http://en.wikipedia.org/wiki/Greedy_algorithm 05 Desember 2012

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, 9 Desember 2011

ttd

Nanda Ekaputra Panjiarga