Penggunaan Algoritma *Greedy* pada Permainan Othello

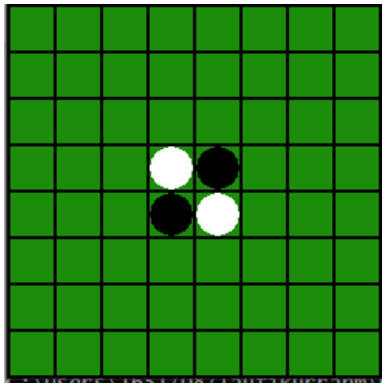
Taufikurrahman Anwar 13517074
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517074@std.stei.itb.ac.id

***Othello* atau terkadang disebut *Reversi* adalah permainan yang dimainkan pada papan berukuran 8x8 petak dan 64 buah keping permainan yang masing-masing memiliki warna yang berbeda pada kedua sisinya (pada umumnya putih dan hitam). Pemenang pada permainan ini adalah pemain yang memiliki keping lebih banyak pada akhir permainan, permainan berakhir ketika semua keping pada papan memiliki warna yang sama atau ketika papan telah penuh terisi. Pada permainan seperti ini hal pertama yang terpikirkan tentunya adalah untuk mendapatkan keping sebanyak mungkin seawal mungkin, karena dengan memiliki lebih banyak keping akan sangat menguntungkan karena memudahkan untuk mendapatkan keping lawan, oleh karena itu makalah ini dibuat untuk mencari tahu apakah teknik "tamak" seperti ini dapat bersaing dengan pemain Othello lainnya.**

Kata kunci—tamak; Othello; bersaing; Reversi; papan; permainan; pemain.

I. PENDAHULUAN

Permainan Othello adalah permainan yang pertama kali dikembangkan sebagai permainan milik Nintendo pada tahun 1980 oleh Goro Hasegawa, permainan ini dikembangkan berdasarkan permainan papan bernama *Reversi* yang dibuat oleh antara Lewis Waterman atau John W. Mollett (keduanya saling menuduh meniru satu sama lain) pada tahun 1883.



Gambar 1.1 Posisi awal papan Othello

Permainan Othello dimainkan pada sebuah papan denagn ukuran 8x8 dan 64 buah keping dengan kedua sisi masing-masing koin memiliki warna yang berbeda (untuk kasus ini hitam dan putih), permainan dimulai dengan pemain yang memiliki keping yang berada di bagian kiri atas (pada gambar 1.1 putih jalan terlebih dahulu) dan kemudian saling bergantian mengambil langkah. Permainan berakhir ketika papan telah terisi penuh atau salah satu pemain sudah tidak memili keping pada papan, saat permainan berakhir pemain dengan jumlah keping terbanyak yang menjadi pemenangnya.

Pemain hanya dapat meletakkan kepingnya pada petak yang berdekatan dengan keping yang lain dan pemain dapat mengambil alih keping musuh dengan mengapitnya diantara satu keping miliknya dan satu keping yang baru saja diletakkan (mengambil alih keping akan mengubah sisi keping sehingga menjadi milik pemain lain), sifat permainan ini yang membuat algoritma *greedy* menjadi pilihan untuk makalah ini, karena untuk setiap langkah pemain akan lebih diuntungkan apabila pemain tersebut memiliki banyak keping untuk mengapit keping lawan dan menambah keping sendiri.

II. DASAR TEORI

A. *Algoritma Greedy*

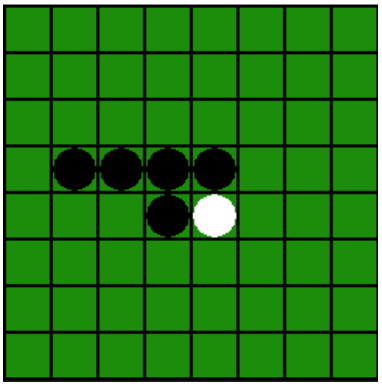
Algoritma *Greedy* adalah salah satu algoritma yang sering digunakan untuk melakukan optimasi, karena algoritma *greedy* ini terhitung cepat dibandingkan algoritma seperti *Brute-Force* atau *Depth-First-Search* yang akan menelusuri semua kemungkinan yang ada untuk suatu masalah. Algoritma *Greedy* idenya adalah untuk mencari nilai optimal lokal dan berharap agar nilai tersebut dapat membawa ke arah optimal global, hal ini berarti algoritma ini sangatlah tidak akurat dan tidak selalu menghasilkan nilai optimal.

Pada permainan Othello ini, sifat algoritma *greedy* yang cenderung tidak akurat ini dapat ditoleri, satu hal karena ini hanya permainan, hal lain karena pada setiap langkah kita hendaknya mencari sebanyak mungkin keping tambahan untuk membantu kita pada langkah-langkah berikutnya, hal inilah yang diharapkan bisa membawa komputer kita ke arah optimum global (kemenangan).

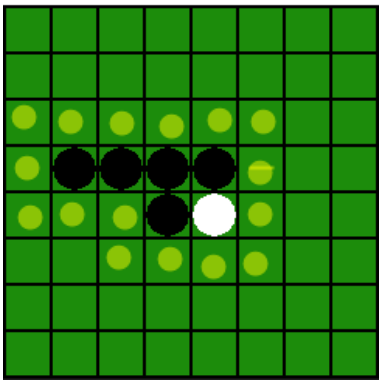
B. *Himpunan Kandidat*

Himpinan Kandidat pada algoritma *greedy* adalah kumpulan dari elemen yang akan menjadi pertimbangan dalam algoritma untuk menentukan pilihan (umumnya dipilih yang terkecil atau terbesar). Elemen yang dapat disebut sebagai anggota himpunan kandidat adalah elemen yang tidak melanggar batasan dari permasalahan (contoh : pada *Knapsack Problem* terdapat batasan untuk tidak melewati batas bobot yang terdapat untuk tas).

Himpunan Kandidat dalam algoritma *greedy* untuk permainan ini adalah petak-petak yang dapat ditempati oleh keping milik komputer dengan mengikuti peraturan yang berlaku pada permainan Othello.



Gambar 2.1 Contoh kondisi papan Othello

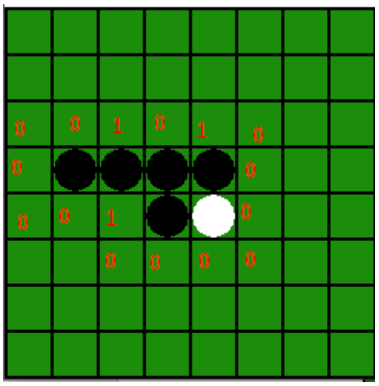


Gambar 2.2 Petak-petak kandidat

Untuk contoh pada gambar 2.1 petak yang menjadi anggota himpunan kandidat adalah petak yang berdekatan dengan keping lain (pada gambar 2.2 petak dengan lingkaran kuning adalah petak-petak yang menjadi anggota himpunan kandidat). Untuk variasi permainan ini yang digunakan adalah variasi dimana pemain tidak harus mengambil keping lawan di setiap langkahnya.

C. *Fungsi Seleksi*

Fungsi seleksi dalam Algoritma *Greedy* adalah fungsi yang menentukan kandidat mana yang akan dipilih dari semua anggota himpunan kandidat. Untuk kasus permainan ini, komputer akan mnelusuri semua anggota himpunan kandidat dan menentukan posisi mana yang paling menguntungkan untuk diambil (pada kasus ini posisi yang membuat komputer memperoleh banyak keping maksimum).



Gambar 2.3 Nilai fungsi seleksi untuk setiap petak kandidat

Pada gambar 2.3 dapat dilihat bahwa terdapat banyak petak yang memiliki hasil fungsi seleksi yang sama, hal ini akan kita abaikan dan kita hanya akan mengambil nilai yang terakhir kali muncul saat pencarian, tentunya hal ini tidak optimal untuk bermain secara serius karena posisi petak dalam permainan ini juga krusial, tetapi pada algoritma *greedy* ini kita hanya melihat nilainya dari jumlah keping yang bisa didapatkan maka ketiga petak dengan nilai '1' akan dianggap sama.

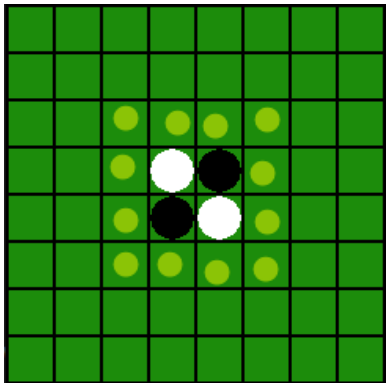
III. Algoritma dan Implementasi Kode

Untuk masalah ini, saya membuat program permainan Othello dengan lawan komputer yang berpikir secara *greedy*. Program ditulis dengan bahasa *python* dengan memanfaatkan pustaka *pygame* untuk antar-muka pemain.

Pada awal program perlu diinisialisasi kondisi petak, jumlah keping yang berada di papan dan himpunan kandidat awalnya.

```
board = [
    [0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0],
    [0,0,0,1,2,0,0,0],
    [0,0,0,2,1,0,0,0],
    [0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0],
]
valid_tile = [
    [2,2],[2,3],[2,4],[2,5],
    [3,2],[3,5],
    [4,2],[4,5],
    [5,2],[5,3],[5,4],[5,5]
]
piece = [2,2]
```

Gambar 3.1 Implementasi kondisi papan dalam program



Gambar 3.2 Kondisi papan sebenarnya

A. *Himpunan Kandidat*

Himpunan Kandidat dalam program ini diimplementasikan sebagai sebuah larik dengan elemen unik, hal ini membantu untuk agar tidak mengolah petak yang sama dua kali dan tidak perlu mencari lagi petak mana yang sah sebagai anggota himpunan kandidat.

Ketika pemain menambahkan keping baru ke dalam permainan, tentunya harus ada perubahan pada himpunan anggota kandidat, pertama menghilangkan petak yang telah terpilih, kemudian menambahkan petak baru di sekeliling petak yang terpakai, perubahan tersebut ditangani oleh kode di bawah ini :

```
# menambahkan tile yang baru valid ke valid_tiles
if (bot(x,y,0)) & ([x+1, y] not in valid_tile):
    valid_tile.append([x+1,y])
if (top(x,y,0)) & ([x-1, y] not in valid_tile):
    valid_tile.append([x-1,y])
if (left(x,y,0)) & ([x, y-1] not in valid_tile):
    valid_tile.append([x,y-1])
if (right(x,y,0)) & ([x, y+1] not in valid_tile):
    valid_tile.append([x,y+1])
if (botleft(x,y,0)) & ([x+1, y-1] not in valid_tile):
    valid_tile.append([x+1,y-1])
if (topright(x,y,0)) & ([x-1, y+1] not in valid_tile):
    valid_tile.append([x-1,y+1])
if (topleft(x,y,0)) & ([x-1, y-1] not in valid_tile):
    valid_tile.append([x-1,y-1])
if (botright(x,y,0)) & ([x+1, y+1] not in valid_tile):
    valid_tile.append([x+1,y+1])
```

Gambar 3.3 Implementasi perubahan himpunan Kandidat

B. *Fungsi Seleksi*

Untuk fungsi seleksi, seperti yang dijelaskan sebelumnya bahwa fungsi seleksi hanya menelusuri anggota himpunan kandidat dan akan memilih salah satu dari anggota himpunan kendidat yang menghasilkan nilai tertinggi untuk menjadi solusi.

```
def greed(screen):
    tmp = []
    mak = 0
    for tile in valid_tile:
        point = (
            greedright(tile[0],tile[1]) +
            greedleft(tile[0], tile[1]) +
            greedtop(tile[0], tile[1]) +
            greedbot(tile[0], tile[1]) +
            greedtopleft(tile[0], tile[1]) +
            greedtopright(tile[0], tile[1]) +
            greedbotleft(tile[0], tile[1]) +
            greedbotright(tile[0], tile[1])
        )
        if point >= mak:
            tmp = tile
            mak = point
    if click_tile(tmp, 0, screen) != None:
        return 0
```

Gambar 3.4 Implementasi Kode Fungsi Seleksi

Seperti yang dapat dilihat, algoritma akan mencari ke delapan tetangga petak kandidat untuk mengetahui berapa banyak keping yang dapat diperoleh dengan meletakkan keping di petak tersebut dan kemudian akan memilih petak yang memiliki nilai tertinggi (dengan fungsi click_tile).

C. *Akhir Permainan*

Permainan Othello akan berakhir ketika terjadi salah satu dari hal berikut :

- Papan Penuh (tidak bisa melanjutkan permainan)
- Salah satu pemain kehabisan keping pada papan (apabila tetap dipaksa bermain, lawan akan selalu bisa mengambil alih keping yang baru diletakkan pada bagian selanjutnya, yang memaksa pemain yang kehabisan keping sudah pasti kalah.)

Kondisi tersebut diimplementasikan dalam bentuk validasi status permainan seperti kode di bawah ini :

```
if(valid_tile == []):
    if (piece[0] > piece[1]):
        print("WHITE WINS!")
        return 0
    elif(piece[0] < piece[1]):
        print("BLACK WINS!")
        return 0
    else:
        print("IT'S A DRAW!")
        return 0
elif (piece[0] == 0):
    print("BLACK WINS!")
    return 0
elif (piece[1] == 0):
    print("WHITE WINS!")
    return 0
```

Gambar 3.5 Kondisi akhir permainan

Dari kode di atas ada beberapa hal yang perlu diperhatikan :

- Tidak mungkin ada kasus permainan berakhir seri ketika masih ada petak yang bisa ditempati
- Petak valid hanya akan kosong ketika seluruh petak pada papan telah penuh terisi

- Kita cukup membandingkan jumlah keping masing-masing ketika papan penuh

Dengan menyimpan status permainan dengan larik seperti ini, kita dapat menghemat waktu karena tidak perlu mencari berulang-ulang setiap kondisi permainan berubah.

IV. Analisis

Secara keseluruhan, implementasi kode seperti di atas sudah cukup untuk menjadi lawan pemain biasa. Hal ini juga menjadi kelebihan Algoritma *Greedy*, karena idenya cukup sedehana maka mudah untuk menuliskannya menjadi sebuah program dan tidak akan menjadi terlalu panjang.

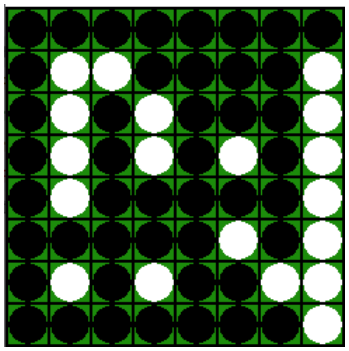
Untuk mengetahui bagaimana performa dari algoritma *greedy* untuk permainan ini, saya menyiapkan lingkungan percobaan untuk melawan manusia dengan antarmuka seperti contoh gambar sebelum-sebelumnya dan sebuah lingkungan lagi untuk melawan komputer lain yang mengambil langkah secara acak, untuk lingkungan untuk melawan komputer acak tidak diberi antarmuka sama sekali dan hanya diambil hasil pertandingannya

A. *Uji Melawan Manusia*

Pada Uji Melawan Manusia, saya menjadi sampel uji. Saya menjadi sampel uji karena saya tidak begitu berpengalaman bermain permainan ini, jadi saya merasa saya adalah representatif yang baik untuk orang awam.

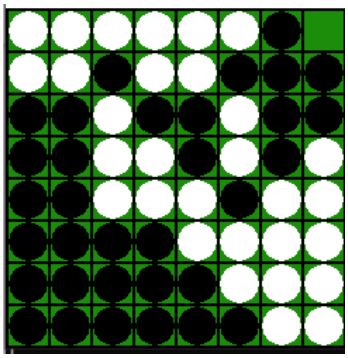
Percobaan dilakukan sebanyak 5 kali dengan manusia sebagai keping putih dan komputer sebagai keping hitam dengan hasil sebagai berikut :

- Percobaan 1



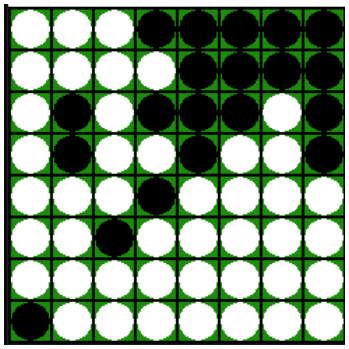
Gambar 4.1 Komputer menang pada permainan 1

- Percobaan 2



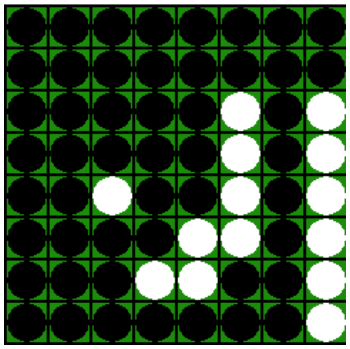
Gambar 4.2 Permainan 2 dimenangkan komputer (petak pada pojok kiri atas diisi oleh keping hitam, tetapi program berhenti sebelum sempat meletakkan keping hitam di petak tersebut)

- Permainan 3



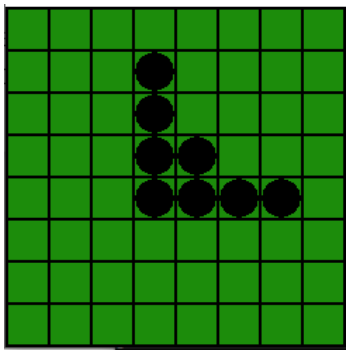
Gambar 4.3 Permainan 3 dimenangkan oleh manusia

- Permainan 4



Gambar 4.4 Permainan 4 dimenangkan oleh komputer

- Permainan 5



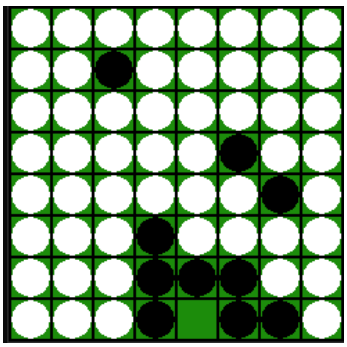
Gambar 4.5 Permainan 5 berkahir awal dengan dimenangkan oleh komputer

B. *Uji Melawan Komputer Lain secara Daring*

Pengujian kurang lebih dilakukan sama seperti pengujian dengan manusia, tetapi kali ini manusia akan mengikuti langkah dari komputer lain.

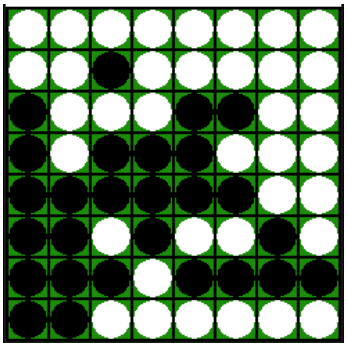
Percobaan ini dilakukan hanya 3 kali pada pengaturan kesulitan paling rendah dengan hasil sebagai berikut (komputer lain sebagai keping putih dan komputer *greedy* sebagai keping hitam):

- Percobaan 1



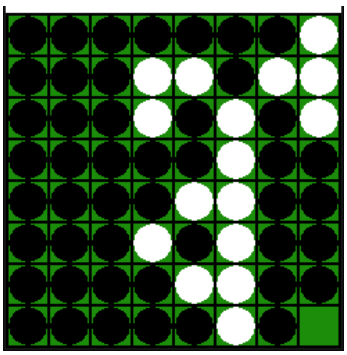
Gambar 4.6 Komputer lain menang pada permainan 1

- Percobaan 2



Gambar 4.7 Komputer lain menang pada permainan 2

- Percobaan 3



Gambar 4.8 Komputer *Greedy* menang pada permainan 3

C. Percobaan Melawan Pemain Acak

Percobaan ini dilakukan dengan mengulang permainan acak melawan komputer *Greedy* dengan mencatat semua hasilnya.

```
def ran():
    tile = random.choice(valid_tile)
    if click_tile(tile, 1) != None:
        return 0
```

Gambar 4.9 Implementasi kode untuk pemain acak

Percobaan ini dilakukan sebanyak 6025 kali dengan hasil sebagai beriut :

- *Greedy* menang sebanyak 5406 kali
- *Random* menang sebanyak 532 kali
- Terjadi seri sebanyak 87 kali

KESIMPULAN

Algoritma ini tentunya masih sangat naif, karena yang menjadi pertimbangan hanyalah jumlah keping yang dapat diperoleh pada petak tertentu, algoritma ini masih bisa dikembangkan dengan menambahkan pertimbangan untuk posisi petak (semisal, petak yang berada lebih luar akan bernilai lebih tinggi).

Dari hasil percobaan dapat dilihat mulai dari percobaan melawan manusia, hasilnya hanya sekali manusia menang melawan komputer *greedy* dari 5 kali percobaan, hal ini menunjukkan bahwa algoritma ini masih lebih baik daripada orang awam ketika bermain permainan Othello.

Pada pecobaan melawan komputer lain, dapat jelas terlihat bahwa strategi *greedy* ini masih belum sebaik itu, karena bahkan pada tingkat kesulitan terendah komputer *greedy* masih kalah 2 dari 3 kali percobaan.

Pada percobaan melawan pemain acak, sangat terjadi ketimpangan, dengan komputer *greedy* menang untuk sekitar 90% dari total permainan, hasil ini menunjukkan bahwa memilih petak dengan acak bukanlah cara terbaik untuk menentukan pilihan.

Percobaan melawan manusia diniatkan untuk merepresentasikan garis tengah antara buruk dan baik, dan percobaan melawan komputer lain diniatkan untuk merepresentasikan garis atas sebagai pemain yang baik, dan pemain acak ada untuk merepresentasikan pemain yang buruk. Dari hasil percobaan diatas tentunya masih bisa lebih banyak percobaannya agar datanya lebih akurat, tetapi untuk saat ini algoritma *greedy* bisa dibilang berada di atas rata-rata dan merupakan cara yang cukup simpel untuk dicoba oleh manusia dengan kesempatan menang yang cukup tinggi. Tetapi algoritma ini tidak bisa menjamin untuk selalu memenangkan permainan Othello.

REFERENCES

[1] *"Brief History of Othello"*. Othello Museum.

[2] *"Strategi Algorima"* . Rinaldi Munir

[3] *"Introduction to Algorithms"* . (Cormen, Leiserson, Rivest, and Stein)

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, 26 April 2019

Taufikurrahman Anwar dan 1351707