Penerapan Algoritma Greedy dalam Permainan Hangman

Ariel Herfrison – 13522002[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): [1]13522002@std.stei.itb.ac.id

***Abstract*—Permainan Hangman adalah salah satu permainan paling populer di dunia. Seperti banyak permainan atau permasalahan lain, penyelesaian permainan Hangman dapat diotomatisasi menggunakan algoritma; salah satunya adalah algoritma greedy. Pembahasan ini menguji performa algoritma greedy menggunakan teknik heuristic berdasarkan jumlah kemunculan huruf diantara semua kata dan berdasarkan jumlah kemunculan suatu huruf bersama suatu huruf lain. Pembahasan menemukan bahwa tidak ada perbedaan performa yang signifikan diantara kedua teknik heuristic tersebut. Pembahasan juga menemukan bahwa algoritma greedy belum cukup efektif dalam menyelesaikan permainan Hangman karena algoritma greedy cenderung membutuhkan lebih dari 7 tebakan untuk menebak kata dengan benar. Hal tersebut menunjukkan pentingnya implementasi teknik heuristic yang baik untuk menyelesaikan permainan Hangman dengan efektif.**

Kata Kunci—Hangman; Greedy; Heuristic; Tebak; Kata

I. PENDAHULUAN

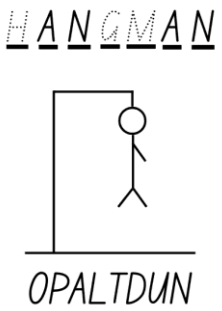
Permainan teka-teki adalah salah satu kategori permainan paling populer di dunia. Salah satu aspek yang membuat teka-teki sangat menarik adalah bagaimana teka-teki dapat merangsang otak kita; menantang kita untuk berpikir di luar kotak untuk memecahkan beragam banyak masalah yang disediakannya. Salah permain teka-teki tersebut adalah Hangman. Cara kerja permainan Hangman cukup sederhana. Pemain akan diberikan sebuah kalimat atau kata yang tidak diketahui dan pemain harus menebak kalimat atau kata tersebut huruf per huruf. Namun, pemain hanya memiliki jumlah tebakan yang terbatas, biasanya 7 tebakan salah, untuk menebak kalimat atau kata tersebut.

Meskipun salah satu faktor utama yang membuat teka-teki sangat menarik adalah tantangan yang diberikannya, terkadang, antara karena ingin bermain curang atau karena rasa ingin tahu, kita ingin mencari cara mudah untuk menyelesaikan teka-teki. Salah satu cara tersebut adalah dengan melakukan otomatisasi pencarian solusi menggunakan algoritma. Terdapat banyak algoritma pemecahan masalah di dunia ilmu komputer. Salah satu algoritma tersebut adalah Algoritma Greedy. Algoritma Greedy bergantung pada teknik heuristic yang digunakannya untuk menentukan solusi permasalahan. Oleh karena itu, dapat dibuat berbagai algoritma greedy dengan teknik heuristic yang berbeda.

II. DASAR TEORI

A. *Hangman*

Hangman adalah sebuah permainan tebak kata dimana pemain harus menebak sebuah kalimat dengan jumlah tebakan yang terbatas. Pemain harus menebak kalimat tersebut secara huruf per huruf. Apabila huruf yang ditebak tidak terdapat di dalam kalimat, tebakan tersebut dikatakan sebagai tebakan salah. Apabila pemain telah melakukan 7 kali tebakan salah, pemain akan kalah permainan. Dalam pembahasan ini, permainan Hangman hanya akan menggunakan kata; bukan kalimat maupun frasa. Kata-kata yang digunakan juga akan berbahasa Inggris dan heuristic yang digunakan juga akan menggunakan kamus bahasa Inggris sebagai basisnya. Permainan juga akan tetap dilanjutkan apabila sudah terdapat 7 tebakan salah, tetapi jumlah tebakan salah akan diperhitungkan.



Gambar 1. *Snapshot* Permainan Hangman

B. *Algoritma Greedy*

Algoritma greedy adalah salah satu algoritma paling sederhana dan paling umum untuk menyelesaikan suatu masalah. Algoritma greedy mirip dengan algoritma brute force, hanya saja, algoritma greedy menggunakan suatu *heuristic* untuk menentukan langkah yang diambil selanjutnya dalam suatu tahap. *Heuristic* tersebut adalah fungsi yang "harapannya" membawa algoritma menuju solusi yang optimal. Oleh karena itu, algoritma greedy tidak dapat melihat konsekuensi yang ditimbulkan dari suatu pilihan terhadap tahap ke depannnya. Algoritma greedy juga memiliki sebuah masalah yang cukup besar dan umum, yaitu bahwa algoritma greedy dapat menimbulkan *infinite loop* karena terjebak dalam pemilihan optimum lokal.

Secara umum, algoritma greedy terdiri dari 6 elemen. Keenam elemen tersebut adalah himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi obyektif. Himpunan kandidat adalah semua pilihan yang dapat diambil dalam suatu tahap. Dalam kasus permainan Hangman, himpunan kandidat adalah 26 huruf yang ada di dalam alfabet. Himpunan solusi adalah himpunan yang berisi semua kandidat yang telah terpilih. Dalam kasus Hangman, ini berarti huruf-huruf di dalam kalimat atau kata yang berhasil ditebak oleh pemain. Fungsi solusi adalah fungsi untuk memeriksa apakah himpunan solusi sudah memberikan solusi. Dalam kasus permainan Hangman, fungsi ini memeriksa apakah semua huruf di dalam kalimat atau kata sudah berada di dalam himpunan solusi. Fungsi seleksi adalah fungsi untuk menentukan huruf yang akan dipilih selanjutnya dari himpunan kandidat. Fungsi kelayakan adalah fungsi untuk memeriksa apakah kandidat dapat dimasukkan ke dalam himpunan solusi. Dalam kasus permainan Hangman, fungsi ini memeriksa apakah pilihan huruf sudah digunakan untuk menebak. Terakhir, fungsi obyektif adalah fungsi untuk menentukan arah pemilihan fungsi seleksi; antara memaksimumkan atau meminimumkan. Dalam kasus permainan Hangman, algoritma ingin meminimumkan waktu atau jumlah tebakan yang diperlukan untuk menemukan kalimat atau kata.

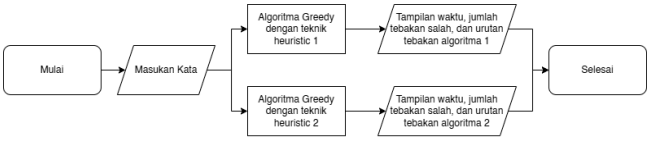
C. *Heuristic*

Heuristic adalah teknik pemecahan masalah yang menjadi panduan algoritma untuk mengambil keputusan. Heuristic tidak sempurna dan belum optimal, tetapi “cukup baik” untuk mengakprosimasi solusi optimal. Dalam algoritma greedy, heuristic digunakan dalam fungsi seleksi untuk menentukan kandidat dari himpunan kandidat yang akan ditambahkan ke dalam himpunan solusi. Dalam kasus permainan Hangman, hal ini berarti menentukan huruf yang akan paling membantu algoritma dalam menebak kalimat. Heuristic yang dapat digunakan bermacam-macam. Beberapa contoh pendekatan pemilihan huruf adalah berdasarkan jumlah kemunculan huruf diantara semua kata atau jumlah kemunculan huruf bersama suatu huruf lain.

III. PEMBAHASAN

A. *Pemodelan Permainan Hangman*

Untuk pembahasan ini, pertama-tama akan dibuat program yang dapat memodelkan permainan hangman. Kalimat yang akan ditebak akan diberikan oleh pengguna sebagai input. Program lalu akan mencoba menebak kalimat tersebut dengan algoritma greedy menggunakan teknik heuristic yang berbeda-beda. Program juga akan menampilkan jumlah tebakan salah yang dilakukan serta urutan tebakan yang dilakukan. Untuk kesederhanaan, program akan berbasis CLI (Command Line Interface). Berikut adalah alur program:



Gambar 2. Alur Program

Berikut adalah contoh tampilan program:

```
Masukkan kata yang akan ditebak: hangman
Algoritma Greedy 1
Jumlah tebakan salah: 11
Urutan tebakan: ['e', 's', 'i', 'a', 'r', 'n', 't', 'o', 'l', 'd', 'u', 'c', 'g', 'p', 'm', 'h']
Algoritma Greedy 2
Jumlah tebakan salah: 11
Urutan tebakan: ['e', 's', 'i', 'a', 'r', 'l', 'n', 'o', 'g', 't', 'u', 'd', 'm', 'c', 'p', 'h']
```

Gambar 3. Tampilan Program

B. *Implementasi Teknik Heuristic*

Di dalam pembahasan ini, akan digunakan 2 teknik heuristic untuk menguji algoritma greedy. Heuristic pertama adalah pemilihan huruf berdasarkan berdasarkan jumlah kemunculan huruf diantara semua kata. Heuristic kedua adalah pemilihan huruf berdasarkan jumlah kemunculan huruf bersama suatu huruf lain. Program akan menggunakan kamus bahasa Inggris sebagai basis perhitungan jumlah kemunculan.

Untuk heuristic pertama, yaitu pemilihan huruf berdasarkan berdasarkan jumlah kemunculan huruf diantara semua kata, program memeriksa apakah suatu huruf muncul setidaknya satu kali di sebuah kata. Apabila huruf tersebut terdapat di dalam kata, maka hal tersebut dihitung sebagai 1 kemunculan dan *count* dari huruf tersebut ditambah satu. Jumlah kemunculan dari setiap huruf akan disimpan dalam sebuah *map* dengan *key* berupa huruf dan *value*-nya adalah jumlah kemunculan/*count* huruf tersebut. Heuristic ini dapat dikembangkan lagi dengan membatasi pemeriksaan pada kata yang panjangnya sama dengan kata yang harus ditebak. Hasil pencarian kemudian dapat diurutkan mulai dari huruf dengan jumlah kemunculan terbanyak. Berikut adalah implementasi heuristic pertama dalam bahasa Python:

```
def heuristic1(listKata, panjangKata):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    countHuruf = {}

    for huruf in alphabet:
        # periksa semua kata dengan panjang = panjangKata
        for kata in listKata[panjangKata]:
            # jika huruf muncul di kata, tambahkan count huruf
            if huruf in kata:
                if huruf in countHuruf:
                    countHuruf[huruf] += 1
                else:
                    countHuruf[huruf] = 1

    # urutkan huruf berdasarkan jumlah kemunculan
    countHurufSorted = dict(sorted(countHuruf.items(), key=lambda item:item[1], reverse=True))

    return countHurufSorted
```

Berikut adalah ilustrasi hasil teknik heuristic 1:

Huruf	Jumlah Kemunculan
e	15210
s	12279
i	10982
a	10785
...	...

Gambar 4. Ilustrasi Hasil Teknik Heuristic 1

Untuk heuristic kedua, yaitu pemilihan huruf berdasarkan jumlah kemunculan huruf bersama suatu huruf lain, untuk setiap kata dan untuk setiap huruf dalam kata tersebut. program akan memeriksa kemunculan huruf lain bersama dengan huruf tersebut. Misal terdapat kata abb, maka jumlah kemunculan huruf a bersama b adalah 1 dan juga sebaliknya; jumlah kemunculan huruf b bersama adalah 1. Jumlah kemunculan dari setiap huruf akan disimpan dalam sebuah *map of map*, yaitu *map* dengan *key* berupa huruf dan *value*-nya adalah *map* lain dengan *key* berupa huruf lain selain huruf tersebut dan *value*-nya adalah jumlah kemunculan huruf tersebut bersama huruf lain tersebut. *Map* ini juga dapat dilihat sebagai matriks simetris dengan isinya berupa jumlah kemunculan huruf pada kolom bersama huruf pada baris. Heuristic ini dapat dikembangkan lagi dengan membatasi pemeriksaan pada kata yang panjangnya sama dengan kata yang harus ditebak. Untuk setiap huruf, hasil pencarian kemudian dapat diurutkan mulai dari huruf lain dengan jumlah kemeunculan terbanyak. Berikut adalah implementasi heuristic kedua dalam bahasa Python:

```
def heuristic2(listKata, panjangKata):
    alphabet = "abcdefghijklmnopqrstuvwxyz"

    # buat map kemunculan huruf bersama huruf lain
    mapHuruf = {huruf:
                {hurufLain:0 for hurufLain in alphabet if hurufLain != huruf}
                for huruf in alphabet}

    # periksa semua kata dengan panjang = panjangKata
    for kata in listKata[panjangKata]:
        hurufUnik = list(set(kata))
        hurufUnik.sort()

        for i in range(len(hurufUnik)):
            huruf1 = hurufUnik[i]
            for j in range(i+1,len(hurufUnik)):
                huruf2 = hurufUnik[j]
                # tambahkan jumlah kemunculan huruf
                mapHuruf[huruf1][huruf2] += 1
                mapHuruf[huruf2][huruf1] += 1

    # urutkan huruf berdasarkan jumlah kemunculan
    for huruf in mapHuruf:
        mapHuruf[huruf] = dict(sorted(mapHuruf[huruf].items(), key=lambda item:item[1], reverse=True))

    return mapHuruf
```

Berikut adalah ilustrasi hasil teknik heuristic 2:

Huruf	Huruf Lain	Jumlah Kemunculan Bersamaan
a	e	6199
	s	5386
	r	4795
	...	...
b	e	2030
	a	1569
	s	1552
	...	...
...	...	...

Gambar 5. Ilustrasi Hasil Teknik Heuristic 2

C. Implementasi Algoritma Greedy

Untuk membantu algoritma, akan dibuat kelas yang akan mensimulasikan permainan Hangman bernama *class Hangman*. Kelas tersebut berfungsi menyimpan kata yang harus ditebak, jumlah tebakan salah yang dilakukan, urutan tebakan yang dilakukan, serta huruf-huruf yang belum ditebak. Kelas tersebut juga memiliki fungsi/*method* untuk mengulangi permainan (*reset*) dan memeriksa tebakan (*checkJawaban*). Fungsi *checkJawaban* akan memiliki beberapa *return value* untuk menandakan hasil tebakan. Fungsi tersebut akan mengembalikan nilai -1 jika tebakan salah, 0 jika tebakan benar tetapi kata belum sepenuhnya ditebak, dan 1 jika tebakan benar dan kata sudah sepenuhnya ditebak. Jika algoritma mencoba menebak huruf yang sudah ditebak, fungsi akan mengembalikan nilai -1 tanpa menambahkan jumlah tebakan

maupun urutan penebakan. Berikut adalah implementasi kelas *Hangman* dalam bahasa Python.

```
class Hangman:
    def __init__(self, kalimat):
        self.kalimat = kalimat
        self.jumlahKesalahan = 0
        self.tebakan = []
        self.sisaHuruf = list(set(kalimat))

    def getPanjangKata(self):
        return len(self.kalimat)

    def reset(self):
        self.jumlahKesalahan = 0
        self.tebakan = []
        self.sisaHuruf = list(set(self.kalimat))

    # returns -1 jika tebakan salah, 0 jika tebakan benar tapi permainan belum selesai, 1 jika tebakan benar dan permainan selesai
    def checkJawaban(self, huruf):
        if huruf in self.tebakan:
            return -1

        self.tebakan.append(huruf)
        if huruf not in self.kalimat:
            self.jumlahKesalahan += 1
            return -1
        else:
            self.sisaHuruf.remove(huruf)

            # jika semua huruf sudah ditebak
            if (len(self.sisaHuruf) == 0):
                print("Jumlah tebakan salah:", self.jumlahKesalahan)
                print("Urutan tebakan:", self.tebakan)
                return 1
            # jika belum semua huruf ditebak
            else:
                return 0
```

Untuk implementasi algoritma greedy dengan teknik heuristic pertama, program akan terlebih dahulu mempersiapkan *map* yang mengurutkan huruf berdasarkan jumlah kemunculannya; mulai dari yang terbanyak. *Map* tersebut didapatkan dari hasil implementasi teknik heuristic pertama yang dijelaskan pada subbab Teknik Heuristic. Program lalu mengiterasi *map* tersebut dan melakukan tebakan mulai dari huruf yang memiliki jumlah kemunculan terbanyak. Iterasi akan berhenti ketika kata berhasil ditebak. Berikut adalah implementasi algoritma greedy menggunakan teknik heuristic 1 dalam bahasa Python:

```
def greedy1(hangman: Hangman, listKata):
    panjangKata = hangman.getPanjangKata()
    mapHuruf = heuristic1(listKata, panjangKata)

    print("Algoritma Greedy 1")
    # iterasi map dari huruf dengan kemunculan terbanyak
    for huruf in mapHuruf:
        result = hangman.checkJawaban(huruf)
        if (result == 1):
            return
```

Untuk implementasi algoritma greedy dengan teknik heuristic kedua, program akan terlebih dahulu mempersiapkan *map* berisi jumlah kemunculan suatu huruf bersama suatu huruf lain yang sudah diurutkan berdasarkan jumlah kemunculan terbanyak. Untuk penebakan pertama atau ketika belum ada huruf yang berhasil ditebak, pemilihan huruf akan menggunakan heuristik pertama (heuristik berdasarkan jumlah kemunculan huruf) karena belum ada huruf yang dapat dijadikan acuan. Ketika ditemukan huruf yang benar, program lalu mengiterasi *map* dari huruf tersebut dan melakukan tebakan mulai dari huruf yang paling sering muncul bersamaan dengannya. Ketika timbul tebakan benar, program akan berhenti mengiterasi *map* dari huruf sebelumnya dan mulai mengiterasi *map* dari huruf yang berhasil ditebak. Iterasi akan berhenti ketika kata sepenuhnya berhasil ditebak. Berikut adalah implementasi algoritma greedy menggunakan teknik heuristic 2 dalam bahasa Python:

```
def greedy2(hangman: Hangman, listKata):
    panjangKata = hangman.getPanjangKata()
    mapHuruf = heuristic2(listKata, panjangKata)
    mapHurufAwal = heuristic1(listKata, panjangKata)

    # fungsi menebak menggunakan heuristic 2
    def tebak(huruf):
        # periksa huruf yang paling sering muncul Bersama huruf yang benar
```

```
        for hurufLain in mapHuruf[huruf]:
            hasilMenebak = hangman.checkJawaban(hurufLain)

            # jika kalimat berhasil ditebak
            if (hasilMenebak == 1):
                return
            # jika tebakan benar tetapi kalimat belum ditebak
            elif (hasilMenebak == 0):
                tebak(hurufLain)
                return

            # jika tebakan salah, tebak dengan huruf selanjutnya

    print("Algoritma Greedy 2")
    # untuk penebakan pertama, penebakan menggunakan heuristic 1
    for huruf in mapHurufAwal:
        result = hangman.checkJawaban(huruf)
        # jika ditemukan huruf benar, gunakan heuristic 2
        if (result != -1):
            tebak(huruf)
            return
```

D. Pengujian

Untuk menguji penerapan algoritma greedy di dalam permainan hangman, akan digunakan beberapa kata sebagai kasus uji, yaitu *notes*, *resin*, *jazz*, *rhythm*, *abruptly*, dan *rhubarb*. Kata *notes* dan *resin* digunakan untuk menguji performa algoritma dalam menebak kata yang menggunakan huruf-huruf yang umum. Kata *jazz* digunakan untuk menguji performa algoritma dalam menebak kata yang tidak menggunakan huruf-huruf yang umum. Kata *rhythm* digunakan untuk menguji performa algoritma dalam menebak kata yang tidak menggunakan huruf vokal. Kata *abruptly* dan *rhubarb* digunakan untuk menguji performa algoritma dalam menebak kata yang relatif sulit. Berikut adalah masil uji masing-masing kata:

1. *Notes*

```
Masukkan kata yang akan ditebak: notes
Algoritma Greedy 1
Jumlah tebakan salah: 4
Urutan tebakan: ['s', 'e', 'a', 'r', 'o', 'i', 'l', 't', 'n']
Algoritma Greedy 2
Jumlah tebakan salah: 4
Urutan tebakan: ['s', 'e', 'r', 'a', 'd', 'l', 't', 'o', 'n']
```

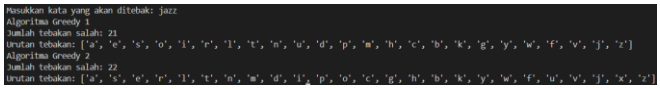
Gambar 6. Uji Coba Kata Notes

2. *Resin*

```
Masukkan kata yang akan ditebak: resin
Algoritma Greedy 1
Jumlah tebakan salah: 4
Urutan tebakan: ['s', 'e', 'a', 'r', 'o', 'i', 'l', 't', 'n']
Algoritma Greedy 2
Jumlah tebakan salah: 2
Urutan tebakan: ['s', 'e', 'r', 'a', 'o', 'i', 'n']
```

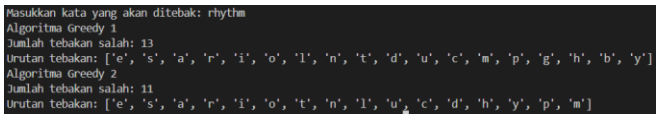
Gambar 7. Uji Coba Kata Resin

3. *Jazz*



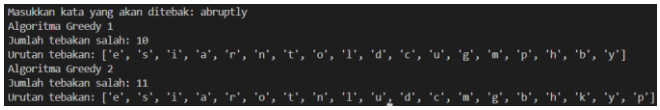
Gambar 8. Uji Coba Kata Jazz

4. *Rhythm*



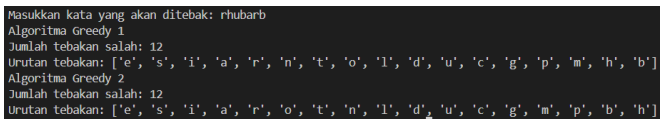
Gambar 9. Uji Coba Kata Rhythm

5. *Abruptly*



Gambar 10. Uji Coba Kata Abruptly

6. *Rhubarb*



Gambar 11. Uji Coba Kata Rhubarb

E. Analisis Pengujian

Berdasarkan hasil pengujian, dapat dilihat bahwa teknik heuristic pertama tidak lebih cepat daripada teknik heuristic kedua. Walaupun teknik heuristic pertama melakukan lebih banyak kesalahan dalam menebak kata *resin* dan *rhythm*, teknik heuristic kedua lebih banyak melakukan kesalahan dalam menebak kata *jazz* dan *abruptly*. Selain itu, teknik heuristic pertama dan kedua juga memiliki jumlah kesalahan yang sama dalam menebak kata *notes* dan *rhubarb*. Hal ini mungkin dikarenakan peggunaan teknik heuristic pertama untuk menebak huruf pertama pada algoritma greedy kedua. Akibatnya, sebagian tebakan pada algoritma kedua sama persis dengan tebakan pada algoritma pertama.

Berdasarkan hasil pengujian, dapat dilihat juga bahwa algoritma greedy belum cukup efektif dalam menyelesaikan permainan hangman. Selain untuk kata *notes* dan *resin*, algoritma greedy membutuhkan lebih dari 7 tebakan untuk menemukan kata yang benar. Pada permainan hangman biasa, hal ini berarti pemain sudah kalah sebelum menemukan kata yang benar. Algoritma greedy juga hanya efektif dalam menebak kata yang menggunakan huruf-huruf yang umum,

seperti kata *notes* dan *resin*. Algoritma greedy sangat tidak efektif dalam menebak kata yang menggunakan huruf-huruf yang tidak umum, seperti kata *jazz*. Penggunaan huruf vokal juga tidak berpengaruh besar pada performa algoritma greedy seperti terlihat pada hasil pengujian kata *rhythm*. Performa algoritma greedy sebagian besar ditentukan oleh seberapa umum huruf yang digunakan oleh kata, seperti terlihat pada hasil pengujian kata *abruptly* dan *rhubarb*. Hal tersebut mungkin dikarenakan teknik heuristic yang digunakan masih relatif sederhana. Teknik heuristic dapat dikembangkan lebih lanjut sehingga memperhatikan posisi huruf yang ditebak atau melakukan tebakan menggunakan huruf yang umum dan huruf yang tidak umum secara bergiliran. Dengan demikian, algoritma greedy mungkin dapat memiliki performa yang lebih baik.

IV. KESIMPULAN

Permainan Hangman dapat diselesaikan dengan algoritma greedy menggunakan teknik heuristic yang berbeda-beda. Teknik heuristic yang digunakan, baik berdasarkan jumlah kemunculan huruf diantara semua kata maupaun jumlah kemunculan huruf bersama suatu huruf lain, tidak terlalu mempengaruhi performa algoritma. Suatu teknik heuristic dapat lebih efektif menebak suatu kata tertentu, tetapi lebih buruk dalam menebak kata yang lain. Walaupun begitu, secara umum, algoritma greedy belum cukup efektif dalam menyelesaikan permainan Hangman. Hal itu dikarenakan algoritma greedy cenderung membutuhkan lebih dari 7 tebakan untuk menebak kata dengan benar. Hal ini berarti pada permainan biasa, pemain akan sudah kalah sebelum menebak kata yang benar. Oleh karena itu, teknik heuristic harus sangat diperhatikan supaya algoritma dapat menebak kata dalam 7 tebakan atau kurang. Teknik heuristic dapat dikembangkan dengan cara memperhatikan posisi huruf yang ditebak atau melakukan tebakan menggunakan huruf yang umum dan huruf yang tidak umum secara bergiliran.

UCAPAN TERIMA KASIH

Segala puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa oleh karena hikmat dan anugerahnya penulis dapat menyelesaikan makalah ini. Penulis menyampaikan terimakasih kepada orang tua penulis yang telah menjadi sumber semangat penulis dalam menempuh perkuliahan dan menyelesaikan makalah ini. Penulis juga menyampaikan terima kasih kepada Dr. Nur Ulfa Maulidevi serta seluruh dosen pengajar mata kuliah IF2211 Strategi Algoritma atas bimbingan dan ilmu yang telah diberikan yang menjadi landasan penulisan makalah ini. Penulis juga berterimakasih kepada teman-teman penulis yang telah mendorong dan menolong penulis dalam menyelesaikan makalah ini.

REFERENSI

[1] 200+ hard hangman words - and how to pick them. (n.d.). https://www.hangmanwords.com/words

[2] Guinness, H. (2022, January 13). *The best starting words to win at Wordle*. Wired. https://www.wired.com/story/best-wordle-tips/

[3] Munir, R. (n.d.). Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/

[4] Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Longman Publishing Co., Inc..

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, 12 Juni 2024

Ariel Herfrison 135220