Optimasi Performa *Word Search Game* Melalui Pemanfaatan Algoritma BFS dan *Bruteforce*

Satria Octavianus Nababan - 13521168
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 13521168@std.stei.itb.ac.id

***Abstract*— *Word Search Game* adalah permainan yang melibatkan pencarian kata di dalam suatu grid berisi huruf-huruf acak. Pencarian kata dapat dilakukan secara horizontal, vertikal, diagonal, dan bahkan secara terbalik. Dalam makalah ini, Penulis membahas pemanfaatan algoritma BFS (*Breadth-First Search*) dan *bruteforce* dalam *Word Search Game*. Dalam makalah ini, kami mengusulkan sebuah pendekatan yang menggabungkan algoritma BFS dan *bruteforce* dalam *Word Search Game*. Penulis menggunakan algoritma BFS sebagai pendekatan awal untuk pencarian kata yang efisien, dan jika kata tidak ditemukan menggunakan BFS, maka pendekatan *bruteforce* digunakan sebagai langkah selanjutnya. Pendekatan ini memungkinkan pemain untuk menikmati keuntungan efisiensi dari algoritma BFS dan juga kemampuan *bruteforce* untuk menangani kasus-kasus yang sulit. Melalui eksperimen dan evaluasi yang kami lakukan, kami menunjukkan bahwa pendekatan yang menggabungkan algoritma BFS dan *bruteforce* dapat meningkatkan efisiensi pencarian kata dalam *Word Search Game*. Dengan pendekatan ini, pemain dapat menemukan kata dengan cepat dan juga menangani kasus-kasus yang mungkin sulit dengan baik.**

Kata kunci: Word Search Game, algoritma BFS, brute force, pencarian kata, grid

I. PENDAHULUAN

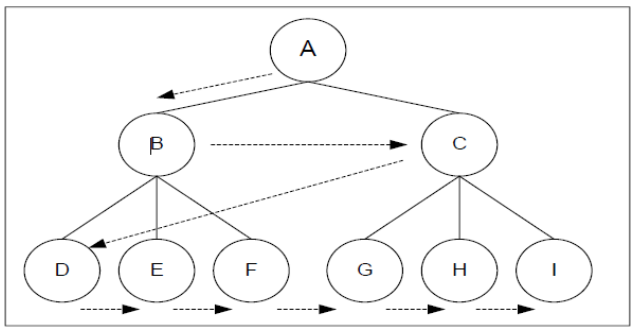
Word Search Game adalah salah satu jenis permainan yang populer di kalangan penggemar teka-teki. Permainan ini melibatkan pencarian kata di dalam sebuah grid yang terdiri dari huruf-huruf acak. Pemain harus menemukan kata-kata yang tersembunyi di dalam grid tersebut dengan menghubungkan huruf-huruf yang saling berdekatan secara horizontal, vertikal, diagonal, dan terbalik. Dalam lingkup *Word Search Game*, ada variasi yang disebut *Animal Word Search Game*. Pada jenis permainan ini, grid berisi huruf-huruf acak yang membentuk kata-kata yang berkaitan dengan nama hewan. Pemain harus mencari dan menemukan kata-kata seperti "gajah," "singa," "jerapah," dan banyak lagi. Tujuan dari permainan ini adalah untuk meningkatkan pemahaman pemain tentang berbagai jenis hewan serta menguji kemampuan mereka dalam mencari kata-kata dengan cepat dan akurat. *Animal Word Search Game* sangat populer di kalangan anak-anak dan juga dapat menjadi tantangan yang menarik bagi pemain dari segala usia. Permainan ini biasanya hadir dalam berbagai tingkat kesulitan, mulai dari tingkat pemula hingga tingkat lanjutan, di mana grid menjadi lebih besar dan kata-kata yang harus ditemukan lebih kompleks. *Animal Word Search Game* juga sering digunakan sebagai alat pendidikan di sekolah-sekolah atau sebagai hiburan dalam buku teka-teki atau aplikasi permainan. *Animal Word Search Game* memberikan kesempatan bagi pemain untuk melatih keterampilan pemecahan masalah, konsentrasi, dan kecermatan visual.

Dalam makalah ini, kami akan membahas pemanfaatan algoritma BFS (*Breadth-First Search*) dan *bruteforce* dalam *Animal Word Search Game*. Algoritma BFS merupakan pendekatan yang efisien untuk mencari kata-kata di dalam grid, sementara *bruteforce* digunakan untuk menangani kasus-kasus yang lebih rumit. Algoritma BFS (*Breadth-First Search*) adalah salah satu algoritma pencarian yang digunakan untuk menjelajahi atau mencari semua simpul dalam sebuah struktur data graf dalam pola "menyapu" secara lebar. Algoritma BFS dimulai dari simpul awal, kemudian mengunjungi semua tetangga langsung dari simpul tersebut, lalu mengunjungi tetangga-tetangga dari tetangga-tetangga tersebut, dan seterusnya, sampai semua simpul yang dapat dicapai dari simpul awal telah dikunjungi. Algoritma BFS sering digunakan untuk mencari jarak terpendek antara dua simpul dalam graf yang tidak terarah atau untuk menjelajahi seluruh simpul dalam graf. Dalam konteks permainan *Word Search*, algoritma BFS dapat digunakan untuk mencari kata-kata dengan efisien dalam grid permainan dengan mempertimbangkan tetangga-tetangga yang mungkin mengandung karakter berikutnya dalam kata yang sedang dicari. Sementara itu, *bruteforce* adalah pendekatan langsung yang memeriksa semua kemungkinan secara eksplisit tanpa memanfaatkan struktur atau metode optimasi tertentu. Dalam konteks *Word Search Game*, *bruteforce* dapat digunakan untuk secara sistematis memeriksa setiap kemungkinan letak kata dalam grid. Pendekatan ini melibatkan memeriksa setiap baris, kolom, dan diagonal dalam grid, serta memeriksa variasi dari arah pencarian, termasuk juga pencarian kata secara terbalik. Kelemahan *bruteforce* terletak pada kompleksitas waktu yang tinggi, terutama ketika grid dan kata yang dicari semakin besar dan kompleks. Meskipun demikian, *bruteforce* memiliki kelebihan dalam menangani kasus yang sulit atau ketika ada lebih dari satu kemungkinan letak kata dalam grid.

II. LANDASAN TEORI

A. Algoritma *Breadth-First Search*

Algoritma *Breadth-First Search* (BFS) adalah salah satu algoritma pencarian graf yang digunakan untuk menjelajahi atau mencari semua simpul dalam sebuah struktur data graf secara sistematis dan berurutan, mulai dari simpul awal dan meluas ke simpul-simpul tetangganya. Sebelum melanjutkan ke simpul-simpul yang lebih jauh. Algoritma BFS melakukan pencarian dalam pola "menyapu" secara lebar, menjelajahi semua simpul pada kedalaman yang sama sebelum melanjutkan ke kedalaman berikutnya. Algoritma BFS dapat diterapkan pada graf yang berarah maupun tidak berarah. Pencarian dimulai dari simpul awal, kemudian dilakukan secara bertahap ke simpul-simpul tetangga yang belum dikunjungi, sebelum melanjutkan ke simpul-simpul tetangga dari tetangga tersebut. Proses ini dilakukan secara berulang sampai semua simpul yang dapat dicapai dari simpul awal telah dikunjungi.



***Gambar 2.1** Breadth first search process.*

Berikut ini adalah langkah-langkah algoritma BFS secara lengkap:

1. Tentukan simpul awal sebagai simpul yang akan dikunjungi pertama kali.
2. Tandai simpul awal sebagai sudah dikunjungi.
3. Buat sebuah antrian kosong (*queue*) dan masukkan simpul awal ke dalam antrian.
4. Selama antrian tidak kosong, lakukan langkah-langkah berikut:
   - Ambil simpul yang berada di depan antrian (simpul yang pertama dimasukkan) dan tandai sebagai simpul yang sedang dikunjungi.
   - Periksa semua tetangga dari simpul yang sedang dikunjungi. Jika tetangga belum dikunjungi, tandai sebagai sudah dikunjungi dan masukkan ke dalam antrian.
   - Setelah selesai mengunjungi semua tetangga, tandai simpul yang sedang dikunjungi sebagai simpul yang sudah dikunjungi.
   - Keluarkan simpul dari antrian (menghapus dari depan antrian).
5. Ulangi langkah 4 sampai antrian kosong.

Selama proses BFS, antrian berperan penting dalam menjaga urutan kunjungan simpul-simpul. Simpul yang dimasukkan lebih awal akan dikunjungi lebih dulu daripada simpul-simpul yang dimasukkan setelahnya. Algoritma BFS menghasilkan pencarian yang "meluas" dari simpul awal, mengunjungi semua simpul pada kedalaman yang sama sebelum melanjutkan ke kedalaman berikutnya. Oleh karena itu, algoritma ini sering digunakan untuk mencari jalur terpendek atau mencari semua simpul yang terhubung pada jarak tertentu dalam graf.

Keuntungan dari algoritma BFS antara lain adalah kemampuannya untuk menemukan jalur terpendek antara dua simpul jika graf tidak memiliki bobot pada setiap sisi, serta kemampuan untuk menemukan semua simpul yang dapat dicapai dari simpul awal. Namun, algoritma ini memiliki kompleksitas ruang yang relatif tinggi, terutama jika graf memiliki banyak simpul dan sisi. Algoritma BFS sangat penting dan banyak diterapkan dalam berbagai aplikasi, termasuk pemodelan jaringan, perencanaan rute, pemrosesan bahasa alami, dan permainan komputer.

B. *Algoritma Bruteforce*

Algoritma *bruteforce* adalah pendekatan sederhana dalam menyelesaikan suatu masalah dengan memeriksa semua kemungkinan secara eksplisit tanpa memanfaatkan struktur data atau metode optimasi tertentu. Dalam konteks pencarian atau pemrosesan data, algoritma *bruteforce* mencoba setiap kemungkinan solusi secara berurutan hingga menemukan solusi yang benar atau mencapai batasan yang ditentukan.

Berikut adalah langkah-langkah umum algoritma brute force:

1. Tentukan semua kemungkinan solusi yang mungkin ada dalam masalah yang ingin diselesaikan.
2. Mulailah dengan solusi awal dalam ruang pencarian.
3. Periksa apakah solusi saat ini memenuhi persyaratan atau kriteria yang diinginkan. Jika solusi ditemukan, proses dapat dihentikan.
4. Jika solusi saat ini tidak memenuhi persyaratan, pindah ke solusi berikutnya dalam urutan yang ditentukan.
5. Ulangi langkah-langkah 3 dan 4 sampai solusi yang memenuhi persyaratan ditemukan atau semua kemungkinan solusi telah dieksplorasi.

Algoritma *bruteforce* cenderung memeriksa semua kombinasi solusi secara berurutan. Ini berarti bahwa kompleksitas algoritma *bruteforce* cenderung tinggi, terutama ketika ruang pencarian sangat besar atau ketika jumlah kemungkinan solusi yang valid sangat besar. Meskipun algoritma *bruteforce* cenderung memiliki kompleksitas waktu yang tinggi, mereka memiliki kelebihan dalam keandalannya dan kesederhanaannya. Algoritma ini dapat diterapkan pada berbagai masalah, terlepas dari struktur data atau karakteristik khusus yang ada.

Namun, dalam beberapa kasus, algoritma *bruteforce* mungkin tidak efisien dan tidak praktis. Misalnya, jika ruang pencarian sangat besar, algoritma *bruteforce* dapat memerlukan waktu yang sangat lama untuk mengeksplorasi semua kemungkinan solusi. Dalam konteks pencarian kata dalam *Word Search Game*, algoritma *bruteforce* dapat diterapkan dengan memeriksa setiap baris, kolom, dan diagonal dalam

grid, serta memeriksa variasi dari arah pencarian, termasuk pencarian kata secara terbalik. Algoritma *bruteforce* akan secara sistematis memeriksa setiap kemungkinan letak kata dalam grid hingga menemukan kata yang dicari atau menjelajahi seluruh ruang pencarian yang mungkin. Pada umumnya, algoritma *bruteforce* digunakan ketika solusi masalah tidak bergantung pada struktur data atau metode optimasi tertentu, dan ketika ukuran ruang pencarian masih terjangkau dalam waktu yang wajar.

C. *Graphical User Interface*

GUI (*Graphical User Interface*) adalah antarmuka pengguna berbasis grafis yang memungkinkan pengguna berinteraksi dengan program atau sistem melalui elemen visual seperti tombol, kotak teks, jendela, dan ikon. GUI menyediakan cara yang lebih intuitif dan mudah digunakan untuk berinteraksi dengan program, dibandingkan dengan antarmuka teks atau baris perintah. GUI memfasilitasi interaksi dengan program melalui aksi pengguna seperti mengklik, mengetik, atau memilih opsi dari elemen-elemen GUI. GUI memungkinkan pengguna untuk berinteraksi dengan program secara visual dan intuitif, menjadikannya lebih mudah digunakan dibandingkan dengan antarmuka berbasis teks.

Komponen GUI adalah elemen-elemen yang membentuk antarmuka pengguna, seperti tombol, label, kotak teks, kotak centang, daftar, dan lain-lain. Setiap komponen GUI memiliki atribut dan metode yang memungkinkan pengaturan tampilan, interaksi, dan manipulasi data yang terkait dengan komponen tersebut. Komponen GUI biasanya ditempatkan dalam hierarki yang terorganisir, seperti jendela, bingkai (*frame*), atau panel, yang memungkinkan penataan dan pengaturan tampilan yang lebih kompleks. Penting bagi sebuah GUI untuk merespons input pengguna dengan cepat dan memberikan umpan balik yang jelas. Interaksi pengguna meliputi aksi seperti mengklik tombol, mengisi kotak teks, memilih opsi dari menu, dan sebagainya. GUI harus mampu menangani interaksi pengguna dengan benar dan memberikan respon yang sesuai.

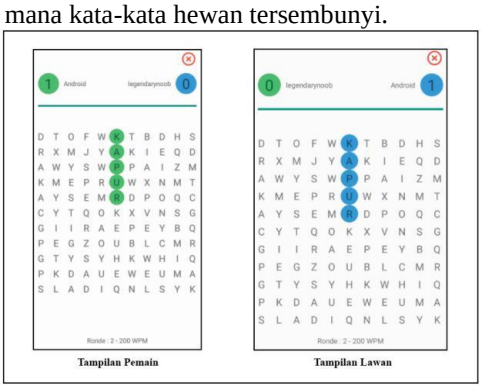
Ada berbagai alat dan *toolkit* yang digunakan untuk membangun GUI, pada makalah ini saya menggunakan *tkinter* sebagai *toolkit* GUI (*Graphical User Interface*) standar untuk bahasa pemrograman Python. *Toolkit* ini menyediakan berbagai komponen dan fungsi untuk membangun antarmuka grafis yang interaktif dan menarik. *Tkinter* dibangun di atas library Tcl/Tk, yang memberikan akses ke widget dan metode untuk mengontrol dan mengatur tampilan antarmuka.

D. *Permainan Pencarian Kata*

Animal Word Search Game adalah permainan teka-teki yang biasanya dimainkan di atas kertas atau melalui aplikasi komputer atau perangkat seluler. Tujuan permainan ini adalah untuk mencari dan mengelompokkan nama-nama hewan yang tersembunyi dalam grid huruf yang terdiri dari huruf-huruf acak. Permainan Animal *Word Search Game* dimainkan di atas grid huruf yang terdiri dari kotak-kotak berisi huruf-huruf acak. Grid huruf ini biasanya berbentuk persegi atau persegi panjang, dan dapat memiliki ukuran yang bervariasi tergantung pada tingkat kesulitan permainan. Di dalam grid huruf, tersembunyi berbagai nama hewan yang harus ditemukan. Nama-nama hewan ini dapat ditemukan dalam berbagai arah, seperti horizontal, vertikal, diagonal, dan terbalik. Biasanya, nama-nama hewan ini memiliki panjang yang berbeda-beda, mulai dari beberapa huruf hingga lebih dari sepuluh huruf.

Gambar 2.2 Ilustrasi Tampilan Awal Game Pencarian Kata

Tugas pemain adalah mencari dan mengelompokkan nama-nama hewan yang tersembunyi dalam grid huruf. Pemain dapat menggunakan pandangan mata mereka untuk mengidentifikasi kata-kata secara visual atau menggunakan metode sistematis seperti membaca grid huruf secara berurutan. Setelah menemukan kata hewan, pemain biasanya menyorot atau menggaris bawahi huruf-huruf yang membentuk kata tersebut. *Animal Word Search Game* dapat memiliki tingkat kesulitan yang bervariasi. Tingkat kesulitan dapat ditentukan oleh ukuran grid huruf, jumlah kata hewan yang harus ditemukan, atau pola dan arah di mana kata-kata hewan tersembunyi.



Gambar 2.3 Ilustrasi Tampilan Game Pencarian Kata Ketika Pemain Berhasil Menebak dan Poinnya Bertambah

Dalam permainan *Animal Word Search Game*, terdapat sistem poin atau skor untuk mengukur keberhasilan pemain. Pemain dapat mendapatkan poin ketika berhasil menemukan kata-kata hewan. Selain itu,

juga seringkali ada pembatasan waktu atau penalti waktu jika pemain tidak dapat menyelesaikan permainan dalam batas waktu tertentu.

III. IMPLEMENTASI

Pada bagian ini, akan dijelaskan bagaimana implementasi program dengan memanfaatkan algoritma BFS (*Breadth-First Search*) dan *bruteforce* dalam perancangan *Word Search Game*. Akan dijelaskan secara rinci langkah-langkah konkret yang diambil dalam mengimplementasikan pemanfaatan algoritma BFS (*Breadth-First Search*) dan *bruteforce* dalam pembuatan *Word Search Game*. Implementasi ini meliputi generasi grid huruf, pencarian kata-kata, penyajian grafis, dan pengoptimalan permainan.

Berikut merupakan pseudocode sederhana dari program yang dibuat :

```
import string
from tkinter import *
import tkinter as tk
import random
from collections import deque

width = 13
height = 13

global score
score = 0

function put_word(word, grid):
   dir = random.choice([[1, 0], [0, 1], [1, 1]])
   x_size = width if dir[0] == 0 else width - len(word)
   y_size = height if dir[1] == 0 else height - len(word)

   x = random.randrange(0, x_size)
   y = random.randrange(0, y_size)

   for i in range(0, len(word)):
      grid[y + dir[1] * i][x + dir[0] * i] = word[i]
   return grid

words = ['TUPAI', 'GAJAH', 'MUSANG', 'ZEBRA', 'MONYET']

grid = create a 2D grid of size height x width filled with random uppercase letters

for word in words:
   grid = put_word(word, grid)

function startwin():
   top = create a new top-level window
   top.title('Word Search Game')
   head = create a label with text "TEMUKAN NAMA HEWAN!!" and add it to top
   for i in range(width):
      lb3 = create a label with text ' '.join(grid[i]) and add it to top

text_input = create a new string variable
sc = create a label with text 'SCORE : ' + str(score) and add it to top

   function btnClick():
      x = get the value from the ans entry

      if bfs(grid, x):
         print "BFS: Kata ditemukan"
         score += 10
         update the text of sc label to 'SCORE : ' + str(score)
      else:
         if x in words:
            print "Bruteforce : Kata ditemukan"
            score += 10
            update the text of sc label to 'SCORE : ' + str(score)
         else:
            print "Kata tidak ditemukan"

      clear the value of text_input

ans = create an entry widget with font, color, and other attributes and assign it to ans variable

check = create a button widget with text "CHECK" and command set to btnClick function

hint0 = create a label with text ">>>>>> Hints <<<<<<"
hint1 = create a label with text "1. Aku sangat besar, tapi aku bukan raja"
hint2 = create a label with text "2. Aku memandang rendah hewan lainnya"
hint3 = create a label with text "3. Aku sangat lincah bergerak di pohon"
hint4 = create a label with text "4. Kotoranku dapat dijadikan luwak"
hint5 = create a label with text "5. Aku suka makan pisang"

arrange the widgets in the top-level window using grid or pack method

function bfs(grid, word):
   queue = create an empty deque

   for i in range(height):
      for j in range(width):
         if grid[i][j] is equal to word[0]:
            add (i, j, 0) to the queue

   while queue is not empty:
      i, j, idx = remove the leftmost item from the queue
```

```
        if idx is equal to len(word) - 1:
            return True

        for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (1, -1), (-1, 1), (-1, -1)]:
            ni, nj = i + dx, j + dy
            if 0 <= ni < height and 0 <= nj < width and grid[ni][nj] is equal to word[idx + 1]:
                add (ni, nj, idx + 1) to the queue

    return False

h = 300
w = 350

root = create the root window
root.title('WordSearch')

canvas = create a canvas widget with height h and width w
frame = create a frame widget with bg color and size

lb1 = create a label with text "Word-Search"
start = create a button with text "START" and command set to startwin function
quit = create a button with text "QUIT" and command set to destroy the root window
lb2 = create a label with text "Word-Search © Tugas Makalah STIMA"

arrange the widgets in the root window using grid or pack method

run the main event loop
```

A. *Representasi dan Generasi Grid Huruf*

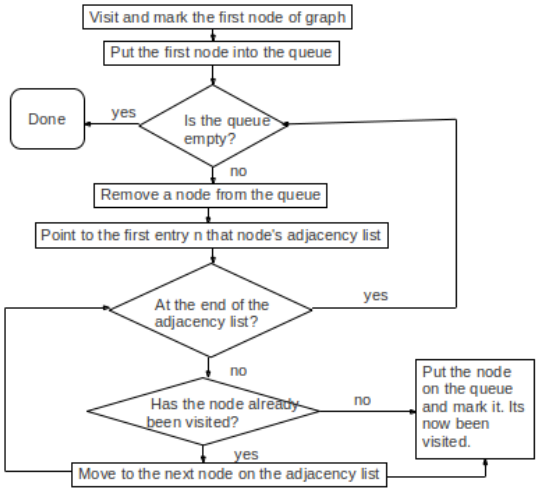
Grid huruf direpresentasikan sebagai matriks dua dimensi. Dalam kode di atas, variabel "grid" merupakan matriks yang terdiri dari karakter-karakter huruf. Setiap elemen matriks merepresentasikan satu kotak pada grid yang berisi huruf. Grid ini digunakan sebagai dasar untuk permainan *Word Search Game*. Proses generasi grid huruf dilakukan dengan menggunakan fungsi "put_word". Fungsi ini mengambil satu kata dari daftar kata-kata yang telah ditentukan sebelumnya. Posisi dan arah kata diatur secara acak pada grid huruf.

Langkah pertama dalam generasi grid huruf adalah memilih arah dan posisi kata. Ini dilakukan dengan memilih secara acak salah satu dari tiga arah (horizontal, vertikal, atau diagonal). Kemudian, posisi awal kata diatur secara acak dalam batasan ukuran grid yang tersedia. Setelah menentukan arah dan posisi kata, huruf-huruf dari kata tersebut ditempatkan pada grid huruf. Setiap huruf diambil secara berurutan dari kata dan ditempatkan pada kotak-kotak yang sesuai dengan arah dan posisi kata tersebut. Proses ini dilakukan dengan menggunakan perulangan untuk mengisi elemen-elemen matriks dengan huruf-huruf dari kata yang diproses. Setelah selesai memproses satu kata, grid huruf yang telah diubah dengan penambahan kata tersebut dikembalikan sebagai output dari fungsi "put_word". Grid huruf ini kemudian digunakan dalam pembuatan permainan *Word Search Game.*

Pada implementasi di atas, kata-kata yang telah ditentukan sebelumnya disimpan dalam daftar "words". Setiap kata dalam daftar tersebut diproses secara berurutan menggunakan fungsi "put_word" untuk menempatkannya pada grid huruf. Setelah semua kata ditempatkan, grid huruf yang dihasilkan digunakan untuk memulai permainan dengan antarmuka grafis menggunakan modul *Tkinter*.

B. *Pencarian Kata dengan BFS*

Salah satu aspek utama dalam *Word Search Game* adalah kemampuan pemain untuk mencari kata-kata dalam grid huruf. Dalam implementasi ini, algoritma BFS digunakan untuk melakukan pencarian kata-kata. Pencarian dimulai dari setiap kotak pada grid yang memiliki huruf pertama dari kata yang dicari. Penulis menggunakan antrian (*queue*) untuk melacak posisi dan indeks kata yang sedang dicari. Algoritma BFS akan menelusuri tetangga-tetangga huruf dalam grid dan membandingkannya dengan huruf berikutnya dalam kata yang sedang dicari. Jika kata berhasil ditemukan, maka permainan akan memberikan umpan balik kepada pemain.



Gambar 3.1 *Flowchart Breadth First Search*

Berikut adalah penjelasan singkat tentang algoritma BFS yang digunakan dalam kode tersebut:

- Pada fungsi bfs(grid, word), pertama-tama kita membuat sebuah *deque* (antrian ganda) yang akan digunakan sebagai antrian untuk menyimpan posisi yang sedang dieksplorasi.
- Kemudian, kita melakukan iterasi melalui setiap sel dalam grid untuk mencari kemungkinan awal dari kata yang dicari. Jika karakter pada sel saat ini sama dengan karakter pertama dari kata, maka kita menambahkan posisi sel tersebut ke dalam antrian dengan indeks karakter saat ini disetel ke 0.

- Selanjutnya, kita mulai menjalankan algoritma BFS. Selama antrian tidak kosong, kita mengambil elemen terdepan dari antrian (yang merupakan posisi yang sedang dieksplorasi).
- Jika indeks karakter saat ini adalah indeks terakhir dari kata (panjang kata dikurangi 1), berarti kita telah menemukan kata secara keseluruhan dalam grid. Dalam hal ini, kita mengembalikan True untuk menandakan bahwa kata ditemukan.
- Jika indeks karakter saat ini bukan indeks terakhir, kita melakukan iterasi melalui semua tetangga yang mungkin (diagonal dan sejajar) dari posisi saat ini. Jika tetangga tersebut berada dalam batas grid dan memiliki karakter yang sama dengan karakter berikutnya dalam kata, maka kita menambahkan tetangga tersebut ke dalam antrian dengan indeks karakter yang diperbarui.
- Jika antrian kosong dan tidak ada kata yang ditemukan, maka kita mengembalikan False untuk menandakan bahwa kata tidak ditemukan.
- Algoritma BFS secara berulang menjelajahi tetangga-tetangga dari posisi saat ini secara melebar (*breadth-wise*) hingga menemukan kata atau telah menjelajahi seluruh grid.

C. *Pencarian Kata dengan Bruteforce*

Algoritma *bruteforce* pada kode di atas dapat ditemukan pada bagian pengecekan kata yang dimasukkan oleh pemain dalam fungsi btnClick(). Berikut adalah penjelasan algoritma brute force tersebut:

- Saat pemain mengklik tombol "CHECK", fungsi btnClick() akan dipanggil.
- Fungsi ini akan mengambil kata yang dimasukkan oleh pemain dari Entry dengan menggunakan ans.get().
- Selanjutnya, kata tersebut akan diperiksa apakah ada di dalam daftar kata yang ada (words). Pengecekan ini dilakukan dengan menggunakan pernyataan if x in words:.
- Jika kata yang dimasukkan oleh pemain ada dalam daftar kata (words), skor pemain akan bertambah 10 poin dengan score += 10. Skor pemain juga akan diperbarui pada label sc yang menampilkan skor.
- Jika kata yang dimasukkan tidak ada dalam daftar kata (words), maka tidak ada perubahan skor yang terjadi.

Algoritma *bruteforce* pada kode tersebut tidak berhubungan dengan algoritma pencarian kata dalam grid. Pengecekan kata yang dimasukkan oleh pemain dilakukan dengan cara membandingkan kata tersebut dengan setiap kata dalam daftar words menggunakan operator in. Ini termasuk dalam kategori *bruteforce* karena kita secara sekuensial memeriksa setiap elemen dalam daftar untuk mencari kecocokan.

D. *Penyajian Grafis dan Antarmuka Pengguna*

Agar permainan lebih menarik dan mudah digunakan, Penulis mengimplementasikan antarmuka grafis menggunakan *library* atau *framework* GUI menggunakan modul *tkinter* pada Python.. Antarmuka pengguna menyediakan grid huruf yang ditampilkan dengan huruf-huruf yang sesuai dengan hasil generasi grid. Daftar kata-kata yang harus dicari juga ditampilkan kepada pemain. Pemain dapat mengklik huruf-huruf untuk membentuk kata dan mendapatkan umpan balik jika kata tersebut ditemukan.

Berikut adalah penjelasan implementasi GUI pada program :

- Pertama, kita mengimpor modul tkinter dengan menggunakan pernyataan *from tkinter import* * dan *import tkinter as tk*. Ini meng-*import* seluruh fungsi dan kelas yang diperlukan dari modul tkinter.
- Kemudian, kita mendefinisikan dimensi lebar (*width*) dan tinggi (*height*) untuk grid permainan kata.
- Selanjutnya, kita mendefinisikan variabel *global score* dan menginisialisasinya dengan nilai 0.
- Fungsi put_word(word, grid) digunakan untuk menempatkan kata secara acak di dalam grid permainan. Ini memilih secara acak arah kata (horizontal, vertikal, atau diagonal), dan mengatur posisi kata di dalam grid.
- Daftar kata-kata yang akan dicari (*words*) diinisialisasi dengan beberapa kata.
- Grid permainan (grid) diinisialisasi sebagai matriks acak dengan ukuran *width* x *height*. Setiap elemen grid berisi karakter acak dari string huruf kapital.
- *Loop for* digunakan untuk menempatkan setiap kata dalam daftar kata-kata ke dalam grid menggunakan fungsi put_word().
- Fungsi startwin() digunakan untuk membuat jendela GUI utama permainan kata. Ini membuat jendela menggunakan Toplevel() dari modul tkinter dan mengatur judul jendela.
- Pada jendela GUI, ada judul permainan (head) yang ditampilkan menggunakan Label. Setiap baris dari grid permainan ditampilkan dalam bentuk teks menggunakan Label juga.
- Variabel text_input digunakan sebagai variabel StringVar() untuk menyimpan teks yang dimasukkan oleh pemain.
- sc adalah Label yang menampilkan skor saat ini. Skor diperbarui setiap kali kata ditemukan.
- btnClick() adalah fungsi yang dijalankan saat tombol "CHECK" ditekan. Ini mengambil teks yang dimasukkan oleh pemain dari ans (Entry widget) dan memeriksa apakah kata tersebut ada dalam grid menggunakan algoritma BFS (bfs()) dan *bruteforce*.
- Setelah selesai, jendela GUI ditampilkan dengan menggunakan Tk() dari modul tkinter. Jendela utama memiliki judul "*WordSearch*".

- Canvas dengan ukuran yang ditentukan (*height* dan *width*) digunakan untuk menempatkan elemen-elemen GUI.
- *Frame* digunakan untuk mengatur elemen-elemen GUI lainnya seperti judul permainan (lb1), tombol "START" (*start*), tombol "QUIT" (*quit*), dan label penulis (lb2).
- mainloop() digunakan untuk menjalankan jendela GUI dan menangani event-event yang terjadi di dalamnya.

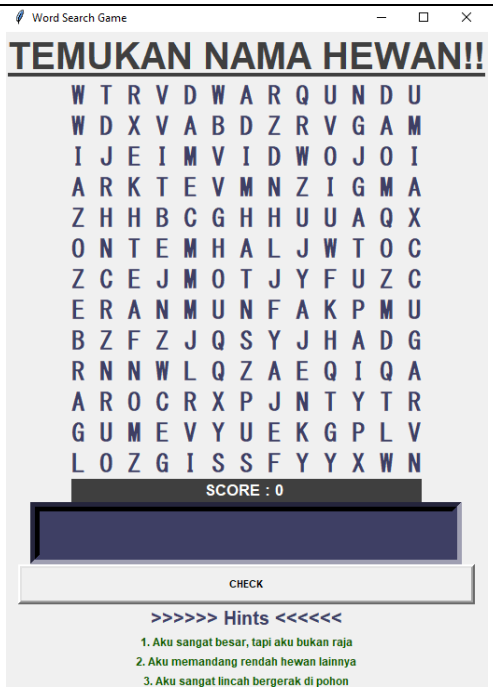
IV. PENGUJIAN

Pada bagian ini akan dilakukan pengujian dari hasil implementasi yang telah diberikan.

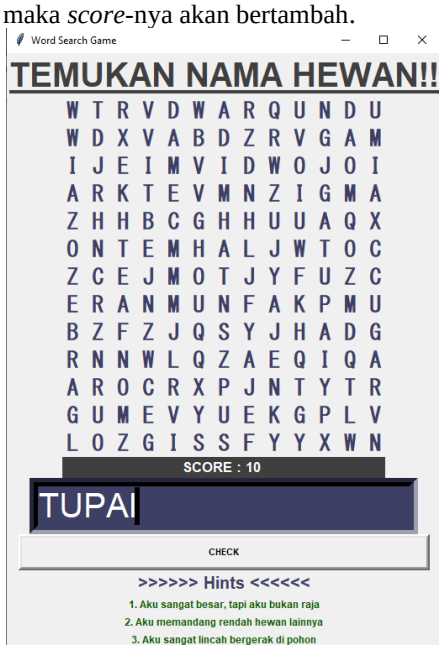
A. *Uji Coba*

<table>
<tr><th>No</th><th>Kasus Uji</th></tr>
<tr><td>1</td><td> 

Gambar 4.1 Tampilan Awal Program

Sesaat setelah menjalankan program, maka akan menampilkan antarmuka tampilan awal program berupa jendela utama yang memiliki judul "WordSearch", dan terdapat tombol "START"), tombol "QUIT" dan label penulis</td></tr>
<tr><td>2.</td><td>Selanjutnya, ketika pengguna memulai permainan dengan melakukan klik pada tombol ”START”, maka akan menampilkan jendela GUI permainan. Terdapat head label yang meminta pengguna menemukan nama hewan dari serangkaian huruf acak yang diberikan. Pengguna juga dapat melihat score dan pada bagian bawah tampilan terdapat petunjuk yang akan membantu pengguna menebak kata hewan yang dicari.</td></tr>
<tr><td></td><td> 

Gambar 4.2 Tampilan GUI Permainan

Pemain dapat mulai memainkan permainan dengan melakukan input tebakannya pada kolom yang tersedia, input harus berupa huruf kapital. Setela melakukan input tebakan, pengguna dapat mengecek tebakannya dengan melakukan klik pada tombol ”CHECk”.</td></tr>
<tr><td>3.</td><td>Ketika pengguna memberikan input tebakan yang benar maka score-nya akan bertambah.
 

Gambar 4.3 Tampilan Ketika Pemain Memberikan Tebakan Yang Benar

”TUPAI” merupakan salah satu nama hewan yang</td></tr>
</table>

	dicari, yang mana kata tersebut ditemukan dengan algoritma BFS. TUPAI BFS: Kata ditemukan 10 ***Gambar 4.4*** Menemukan Kata TUPAI dengan BFS
4.	Jika pemain tidak memberikan input tebakan yang benar, maka *score*-nya tidak akan bertambah.  ***Gambar 4.5*** Pemain Memberikan Tebakan Yang Salah SINGA Kata tidak ditemukan 10 ***Gambar 4.6*** Kata SINGA Tidak Ditemukan
5.	 ***Gambar 4.7*** Pemain menebak Kata GAJAH

	Pemain memberkan tebakan yang benar yang didaptkan dengan algoritma *bruteforce*. GAJAH Bruteforce : Kata ditemukan 50 ***Gambar 4.8*** Kata GAJAH Ditemukan dengan *Bruteforce*

Penulis berhasil melakukan implementasi algoritma yang telah disusun dan mengarsipkannya dalam sebuah tautan yang dapat diakses secara publik. Pengujian lebih lanjut dapat dilakukan dengan mengakses laman github penulis berikut :

https://github.com/satrianababan/Makalah-STIMA.git

B. *Analisis Kompleksitas*

Setelah melakukan pengujian maka dapat dilakukan analisis komplesitas. Kompleksitas merupakan kajian atau studi terhadap sistem kompleks . kata “kompleksitas” berasal dari bahasa latin complexice yang artinya ’totalitas’ atau ’koseluruhan’ sebuah ilmu yang mengkaji totalitas sistem dinamik secara keseluruhan (Dimitrov, 2003). Secara sederhana, dapat dikatakan bahwa sebuah sistem dikatakan kompleks jika sitem itu terdiri dari banyak komponen atau subunit yang saling berinteraksi dan mempunyai prilaku yang menarik, namun, secara bersamaan tidak kelihatan terlaulu jelas jika dilihat sebagai hasil dari interaksi antar sub-unit yang diketahui (Parwani, 2002).

Kompleksitas dapat mengukur waktu yang diperlukan oleh sebuah algoritma dengan menghitung banyaknya operasi/instruksi yang dieksekusi. Untuk mengetahui besaran waktu (dalam satuan detik) untuk melaksanakan sebuah operasi tertentu, maka kita dapat menghitung berapa waktu sesungguhnya untuk melaksanakan algoritma tersebut. Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n. Kompleksitas waktu dibedakan atas tiga macam :

1. Tmax(n) : kompleksitas waktu untuk kasus terburuk (worst case), ¬ kebutuhan waktu maksimum.
2. Tmin(n) : kompleksitas waktu untuk kasus terbaik (best case), ¬ kebutuhan waktu minimum.
3. Tavg(n) : kompleksitas waktu untuk kasus rata-rata (average case) ¬ kebutuhan waktu secara rata-rata

Jumlah operasi perbandingan elemen tabel:

1. Kasus terbaik: ini terjadi bila a1 = x.
   Tmin(n) = 1
2. Kasus terburuk: bila an = x atau x tidak ditemukan.
   Tmax(n) = n
3. Kasus rata-rata: Jika x ditemukan pada posisi ke-j, maka operasi perbandingan (ak = x) akan dieksekusi sebanyak j kali.

$$T_{avg}(n) = \frac{(1+2+3+\ldots+n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$$

Persamaan 4.1 Persamaan Kasus Rata-Rata

Cara lain: asumsikan bahwa P(aj = x) = 1/n. Jika aj = x maka Tj yang dibutuhkan adalah Tj = j. Jumlah perbandingan elemen larik rata-rata:

$$T_{avg}(n) = \sum_{j=1}^{n} T_j P(A[j] = X) = \sum_{j=1}^{n} T_j \frac{1}{n} = \frac{1}{n} \sum_{j=1}^{n} T_j$$

$$= \frac{1}{n} \sum_{j=1}^{n} j = \frac{1}{n} \left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}$$

Persamaan 4.2 Perhitungan Kasus Rata-Rata dengan Elemen Larik [15]

V. KESIMPULAN

Animal Word Search Game adalah permainan di mana pemain harus mencari kata-kata yang terkait dengan hewan dalam grid huruf acak. Tujuan dari permainan ini adalah untuk menemukan dan menggarisbawahi kata-kata hewan yang tersembunyi dalam grid tersebut. Cara bermain *Animal Word Search Game* cukup sederhana. Pemain akan diberikan grid huruf yang terdiri dari beberapa baris dan kolom. Kata-kata hewan akan disembunyikan secara horizontal, vertikal, atau diagonal dalam grid tersebut. Pemain harus mencari dan menggarisbawahi kata-kata tersebut dengan menghubungkan huruf-huruf yang membentuk kata yang tepat.

Makalah ini menggabungkan penggunaan pendekatan BFS dan *bruteforce* dalam perancangan permainan *Animal Word Search*. Pendekatan BFS digunakan untuk mencari kata-kata secara efisien dengan kompleksitas waktu yang rendah. Pendekatan *bruteforce* digunakan sebagai metode cadangan jika BFS tidak menemukan kata yang dicari. Implementasi ini menggabungkan kecepatan dan efisiensi BFS dengan kemampuan *bruteforce* untuk menemukan kata-kata yang mungkin terlewatkan oleh BFS.

VI. UCAPAN TERIMA KASIH

Puji Syukur hanya kepada Tuhan Yang Maha Esa karena hanya atas berkat dan limpahan rahmatNya, penulis dapat menyelesaikan makalah ini dengan baik. Terima kasih juga penulis sampaikan kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. sebagai dosen pengampu dalam mata kuliah IF2211 Strategi Algoritma kelas K02 atas ilmu yang telah diberikan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Tidak lupa terima kasih juga penulis sampaikan kepada orang tua yang senantiasa memberikan dukungan dan motivasi kepada penulis.

LAMPIRAN

Video : ***https://youtu.be/iizu2yVqDvE***

REFERENSI

[1] https://www.gamesver.com/word-search-what-is-it-objective-purpose-and-more/

[2] https://www.youtube.com/watch?v=tiLwW8StyBc

[3] https://elib.unikom.ac.id/files/disk1/705/jbptunikompp-gdl-rizkiprima-35237-9-unikom_r-i.pdf

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf.

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

[6] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

[7] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf

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, 21 Mei 2023

Satria Octavianus Nababan
13521168