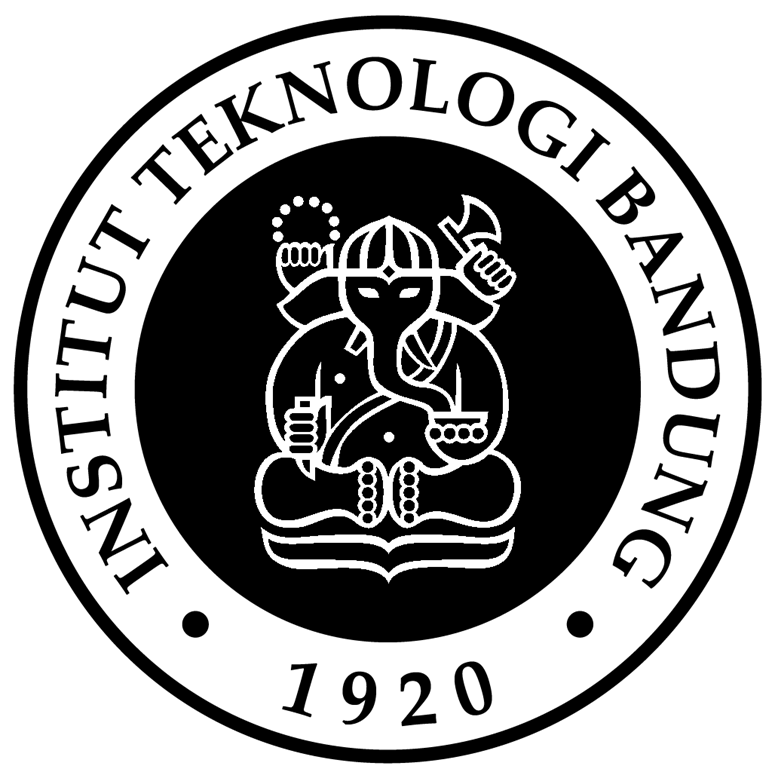
**Laporan Tugas Kecil 3  
 IF2211 Strategi Algoritma**

Aplikasi UCS, A\*, dan Greedy Best First Search untuk Mencari Solusi Terpendek dalam Permainan Word Ladder



Disusun oleh:

Akbar Al Fattah 13522036

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

**Bab 1: Deskripsi Masalah**

*Word ladder* (juga dikenal sebagai *Doublets, word-links, change-the-word puzzles, paragrams, laddergrams*, atau *word golf*) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. *Word ladder* ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai *start word* dan *end word*. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara *start word* dan *end word*. Banyaknya huruf pada *start word* dan *end word* selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.

A screenshot of a game

Description automatically generated

**Gambar 1.** Ilustrasi dan Peraturan Permainan *Word Ladder*

(Sumber: <https://wordwormdormdork.com/>)

Permainannya cukup sederhana bukan? Jika belum paham dengan peraturan permainannya, cobalah untuk memainkan permainannya pada link sumber di atas. Jika sudah paham dengan permainannya, sekarang adalah waktunya kalian untuk membuat sebuah solver permainan tersebut dengan harapan kita dapat menemukan solusi paling optimal untuk menyelesaikan permainan *Word Ladder* ini.

**Bab 2: Dasar Teori**

1. **Algoritma A\***

Algoritma A\* adalah algortima pencarian di graf yang menggunakan jarak antar edge sesungguhnya (g(n)) dan nilai heuristik/estimasi cost (h(n)). Pada algoritma A\*, nilai f(n) adalah f(n) = g(n) + h(n). Nilai heuristik h(n) yang digunakan harus admissible.

Nilai heuristik yang admissible adalah nilai heuristik yang tidak melebihi cost sungguhan g(n). Fungsi untuk menghitung nilai heuristik dibebaskan asal admissible dan dapat diaplikasikan sesuai dengan permasalahan yang ingin diselesaikan.

Berikut adalah langkah-langkah algoritma A\*

1. Buat Priority Queue yang mengutamakan nilai f(n) yang paling kecil.
2. Hitung nilai heuristik h(n) untuk semua node dan g(n) untuk semua edge dan cost g(n) untuk node awal (g(n) = 0 karena g(n) adalah jarak dari node ke titik awal)
3. Hitung nilai f(n) = g(n) + h(n) dari node awal.
4. Masukkan node awal ke priority queue dengan nilai prioritasnya adalah f(n).
5. Ulangi langkah 5 dan seterusnya sampai priority queue kosong atau path sudah ditemukan:
6. Dapatkan node paling depan di priority queue
7. Jika node tersebut sudah pernah dikunjungi, ulangi dari langkah 5, jika tidak, lanjut ke langkah 6.
8. Jika node tersebut tidak sama dengan node tujuan, periksa semua tetangga dari node tersebut dengan langkah di bawah ini:
9. Hitung nilai g(n) yang baru dengan cara mendapatkan nilai g node sekarang ditambah dengan cost dari edge node sekarang ke node neighbor yang sedang diperiksa.
10. Jika nilai g(n) yang baru lebih kecil dari nilai g(n) tetangga yang sedang diperiksa lakukan langkah 10-14. Jika tidak, lewati langkah 10-14,periksa tetangga berikutnya, dan ulangi dari langkah 9.
11. Perbarui nilai g(n) tetangga yang sedang diperiksa dengan g(n) baru yang baru saja dihitung di langkah 8
12. Hitung nilai f(n) = g(n) + h(n) dari node tetangga yang sedang diperiksa.
13. Buat parent dari node tetangga yang sedang diperiksa menjadi node yang sekarang sedang dikunjungi.
14. Masukkan node tetangga tersebut ke priority queue.
15. Jika node yang dikunjungi pada langkah 7 adalah node tujuan, susun path yang ditemukan dan hentikan semua langkah algoritma ini.
16. **Algoritma Greedy Best First Search (GBFS)**

Algoritma Greedy Best First Search adalah algoritma A\* yang memiliki syarat khusus **semua nilai g(n) tidak digunakan (dianggap tidak ada)** dan **h(n) digunakan.** Akibatnya, nilai f(n) untuk Greedy Best First Search adalah **f(n) = h(n).**

Langkah-langkah algoritma GBFS kurang lebih mirip dengan A\* dengan beberapa perbedaan. Berikut adalah langkah-langkah algoritma GBFS.

1. Buat Priority Queue yang mengutamakan nilai f(n) yang paling kecil.
2. Hitung nilai heuristik h(n) untuk semua node
3. Hitung nilai f(n) = h(n) dari node awal.
4. Masukkan node awal ke priority queue dengan nilai prioritasnya adalah f(n).
5. Ulangi langkah 5 dan seterusnya sampai priority queue kosong atau path sudah ditemukan:
6. Dapatkan node paling depan di priority queue
7. Jika node tersebut sudah pernah dikunjungi, ulangi dari langkah 5, jika tidak, lanjut ke langkah 6.
8. Jika node tersebut tidak sama dengan node tujuan, periksa semua tetangga dari node tersebut dengan langkah di bawah ini:
9. Hitung nilai f(n) = h(n) dari node tetangga yang sedang diperiksa.
10. Buat parent dari node tetangga yang sedang diperiksa menjadi node yang sekarang dikunjungi.
11. Masukkan node tetangga tersebut ke priority queue.
12. Jika node yang dikunjungi pada langkah 7 adalah node tujuan, susun path yang ditemukan dan hentikan semua langkah algoritma ini.
13. **Algoritma Uniform Cost Search (UCS)**

Algoritma UCS adalah algoritma A\* yang memiliki syarat khusus **semua nilai h(n) tidak digunakan** dan **g(n) digunakan**. Artinya, UCS tidak menggunakan nilai heuristik h(n) dan tidak perlu mendefinisikan fungsi h(n). Akibatnya, nilai f(n) untuk UCS adalah **f(n) = g(n).**

Langkah-langkah algoritma UCS kurang lebih mirip dengan A\* dengan beberapa perbedaan. Berikut adalah langkah-langkah algoritma UCS.

1. Buat Priority Queue yang mengutamakan nilai f(n) yang paling kecil.
2. Hitung nilai g(n) untuk semua edge dengan g(n) node awal adalah 0
3. Hitung nilai f(n) = g(n) dari node awal.
4. Masukkan node awal ke priority queue dengan nilai prioritasnya adalah f(n).
5. Ulangi langkah 5 dan seterusnya sampai priority queue kosong atau path sudah ditemukan:
6. Dapatkan node paling depan di priority queue
7. Jika node tersebut sudah pernah dikunjungi, ulangi dari langkah 5, jika tidak, lanjut ke langkah 6.
8. Jika node tersebut tidak sama dengan node tujuan, periksa semua tetangga dari node tersebut dengan langkah di bawah ini:
9. Hitung nilai g(n) yang baru dengan cara mendapatkan nilai g node sekarang ditambah dengan cost dari edge node sekarang ke node neighbor yang sedang diperiksa.
10. Jika nilai g(n) yang baru lebih kecil dari nilai g(n) tetangga yang sedang diperiksa lakukan langkah 10-14. Jika tidak, lewati langkah 10-14,periksa tetangga berikutnya, dan ulangi dari langkah 9.
11. Perbarui nilai g(n) tetangga yang sedang diperiksa dengan g(n) baru yang baru saja dihitung di langkah 8
12. Hitung nilai f(n) = g(n) dari node tetangga yang sedang diperiksa.
13. Buat parent dari node tetangga yang sedang diperiksa menjadi node yang sekarang sedang dikunjungi.
14. Masukkan node tetangga tersebut ke priority queue.
15. Jika node yang dikunjungi pada langkah 7 adalah node tujuan, susun path yang ditemukan dan hentikan semua langkah algoritma ini.
16. **Hubungan Antara Algoritma A\*, UCS, Greedy Best First Search, dan Breadth First Search**

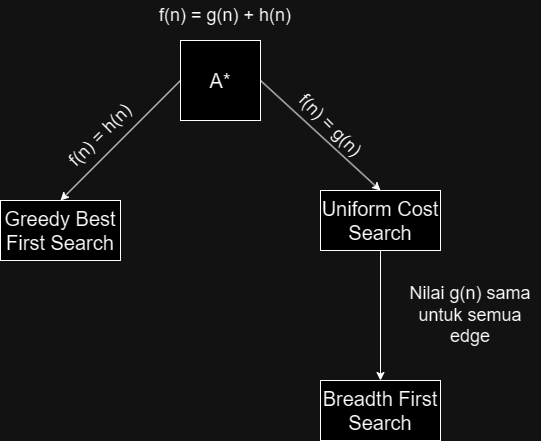
Sebelumnya telah kita ketahui bahwa algoritma A\* adalah bentuk umum dari kedua algoritma lain yang digunakan (UCS dan GBFS). Namun, ada lagi algoritma yang berhubungan dengan ketiga algoritma di atas, yaitu breadth first search (BFS).

Breadth First Search adalah **kasus khusus dari UCS yang memiliki nilai g(n) yang sama semua untuk setiap edge**.

Berikut ini adalah pembuktian bahwa BFS adalah UCS yang memiliki nilai g(n) yang sama semua untuk semua edge.

1. Pada UCS, yang dijadikan nilai prioritas untuk pencarian node yang akan dikunjungi terlebih dahulu adalah nilai **f(n) = g(n).** Nilai g(n) (jarak antar kedua node) bisa sama untuk beberapa atau semua edge dan bisa berbeda dengan edge yang lain. Node yang memiliki nilai g(n) yang sama yang akan dikunjungi terlebih dahulu bergantung pada definisi yang ditetapkan oleh programmer, apakah urutan dimasukkan ke priority queue atau berdasarkan ketentuan lain.
2. Akan tetapi, jika **g(n) bernilai sama semua** **untuk semua edge**, maka **nilai f(n) tidak ada artinya karena tidak ada nilai f(n) yang berbeda.** Akibatnya, nilai prioritas node akan sama semua dan **penggunaan priority queue menjadi tidak berguna di sini** karena **tidak ada node yang diprioritaskan untuk dikunjungi terlebih** dahulu di queue. Selain itu, nilai prioritas node yang sama semua akan menyebabkan node yang dimasukkan ke priority queue adalah **semua node yang merupakan tetangga dari node yang sedang dikunjungi.**
3. **Perhatikan bahwa pernyataan di atas adalah definisi dari Breadth First Search (BFS).**
4. Dari pernyataan nomor 2 dan 3, dapat disimpulkan bahwa **Breadth First Search adalah kasus khusus UCS yang memiliki nilai g(n) yang sama untuk semua edge.**

Berikut adalah gambar hubungan antara A\*, GBFS, UCS, dan BFS.



**Gambar 2.** Hubungan A\*, GBFS, UCS, dan BFS

(Sumber: Diagram yang dibuat sendiri oleh penulis di https://app.diagrams.net/)

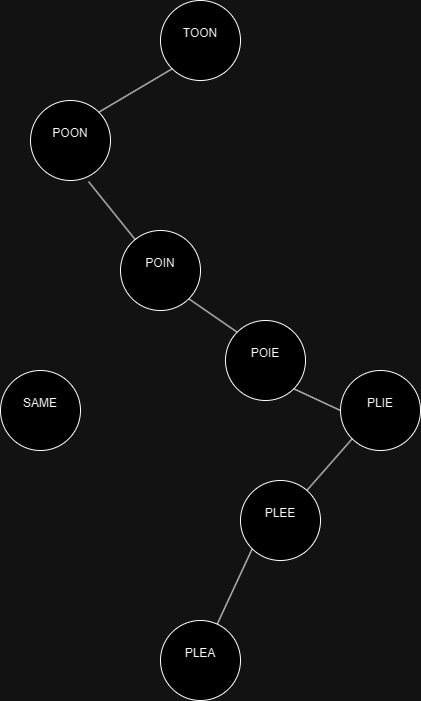
**Bab 3: Pemecahan Masalah**

1. **Konstruksi Graf**

Pada Word Ladder, definisi tetangga dari suatu node adalah **semua kata yang hanya berbeda satu huruf dari node tersebut.**

Contoh ilustrasi: Misalkan kita punya dictionary {TOON, POON, PLEE, SAME, POIE, PLEA, PLIE, POIN}, titik awalnya dimulai dari TOON dan tujuannya adalah PLEA.

Maka, graf yang dikonstruksi untuk memecahkan masalah ini terdapat pada gambar di bawah ini.



**Gambar 3.** Hasil graf yang dibangun dari dictionary

(Sumber: Diagram yang dibuat sendiri oleh penulis di https://app.diagrams.net/)

Hasilnya, didapatkan sebuah graf **unweighted/berbobot sama semua dan undirected/tidak berarah.**

1. **Definisi Nilai h(n) (nilai heuristik graf)**

Nilai heuristik tiap node yang digunakan pada graf ini adalah **banyak huruf yang berbeda dari kata node tersebut dengan tujuan akhir.**

Nilai heuristik ini **admissible** karena estimasi nilai h(n) tidak melebihi dari jarak/cost/g(n) sebenarnya dari setiap node. Nilai **h(n) minimum adalah 0** (semua huruf kata tersebut sama dengan tujuan akhir) dan **nilai h(n) maksimum adalah panjang dari kata tersebut** (semua huruf kata tersebut berbeda dengan tujuan akhir), sedangkan **nilai g(n) minimum adalah 0** dan **nilai g(n) maksimum adalah tak hingga.**

1. **Aplikasi ke tiap algoritma**
2. **UCS**

Perhatikan kembali bahwa graf Word Ladder adalah **graf unweighted/berbobot sama semua.** Akibatnya, ini adalah kasus khusus UCS, yaitu Breadth First Search (BFS). Oleh karena itu, algoritma UCS yang digunakan di graf Word Ladder **ekuivalen dengan algoritma Breadth First Search.** Karena UCS yang digunakan pada adalah BFS, maka UCS **menjamin path yang ditemukan adalah path terpendek.**

1. **GBFS**

Berbeda dengan UCS dan A\*, GBFS **tidak menjamin path yang ditemukan adalah path terpendek** karena GBFS hanya memikirkan nilai heuristik h(n) terkecil dari suatu node tanpa memikirkan jarak node dari titik asal.

1. **A\***

Graf Word Ladder adalah graf unweighted, kita anggap nilai g(n) untuk semua edge adalah 1. Karena h(n) adalah nilai heuristik yang admissible, maka A\* **menjamin path yang ditemukan adalah path terpendek**. Berbeda dengan UCS/BFS, A\* jauh lebih efisien dan lebih cepat karena algoritma A\* **menghindari beberapa node yang dianggap sudah memiliki g(n) yang besar,** sedangkan UCS/BFS mengecek semua tetangga**.**

**Bab 4: Source Code Algortima Pencarian Path**

**Screenshot source code**

**Semua source code ini dapat dilihat di Graph.java**

* Algoritma UCS

A computer screen shot of text

Description automatically generated(Catatan: Implementasi algoritma UCS di problem ini ekuivalen dengan Breadth First Search (BFS) karena graf pada permasalahan ini adalah graf yang tidak memiliki bobot *(unweighted graph),* sudah dijelaskan alasannya sebelumnya)

*Tangkapan Layar Source Code Algoritma UCS (yang ekuivalen dengan BFS)*

* Algoritma Greedy Best First Search

(Catatan: nilai heuristik yang digunakan adalah banyaknya huruf pada suatu kata yang **berbeda** dengan kata **tujuan. Namun, GBFS tidak menjamin bahwa path yang dicari adalah** **path terpendek.**)

A computer screen shot of text

Description automatically generated

*Tangkapan Layar Source Code Algoritma Greedy Best First Search*

* Algoritma A\*

A screen shot of a computer program

Description automatically generated

A blue screen with white text

Description automatically generated

*Tangkapan Layar Source Code Algoritma A\**

**Bab 5: Uji Coba Program**

**Keterangan: dictionary yang digunakan program ini dapat dilihat di situs di bawah ini** https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt

1. EAST -> WEST
2. UCS

A screenshot of a computer

Description automatically generated

1. Greedy Best First Search

A screenshot of a computer

Description automatically generated

1. A\*

A screenshot of a computer

Description automatically generated

1. GUITAR -> PHONES
2. UCS

A screenshot of a computer

Description automatically generated

1. Greedy Best First Search

A screenshot of a computer

Description automatically generated

1. A\*

A screenshot of a computer

Description automatically generated

1. ATLASES -> CABARET
2. UCS

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white background with black text

Description automatically generated

A white background with green text

Description automatically generated

A white background with black text

Description automatically generated

1. Greedy Best First Search

A screenshot of a computer

Description automatically generated

(path di bagian tengah dilewati karena path terlalu panjang)

A screenshot of a computer

Description automatically generated

1. A\*

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white background with black text

Description automatically generated

A white background with green text

Description automatically generated

A white background with black text

Description automatically generated

1. PONY -> SOCK
2. UCS

A screenshot of a computer

Description automatically generated

1. Greedy Best First Search

A screenshot of a computer

Description automatically generated

1. A\*

A screenshot of a computer

Description automatically generated

1. NYLON -> MANGO
2. UCS

A screenshot of a computer

Description automatically generated

A white background with green text

Description automatically generated

1. Greedy Best First Search

A screenshot of a computer

Description automatically generated

A white background with green text

Description automatically generated

A close-up of words

Description automatically generated

1. A\*

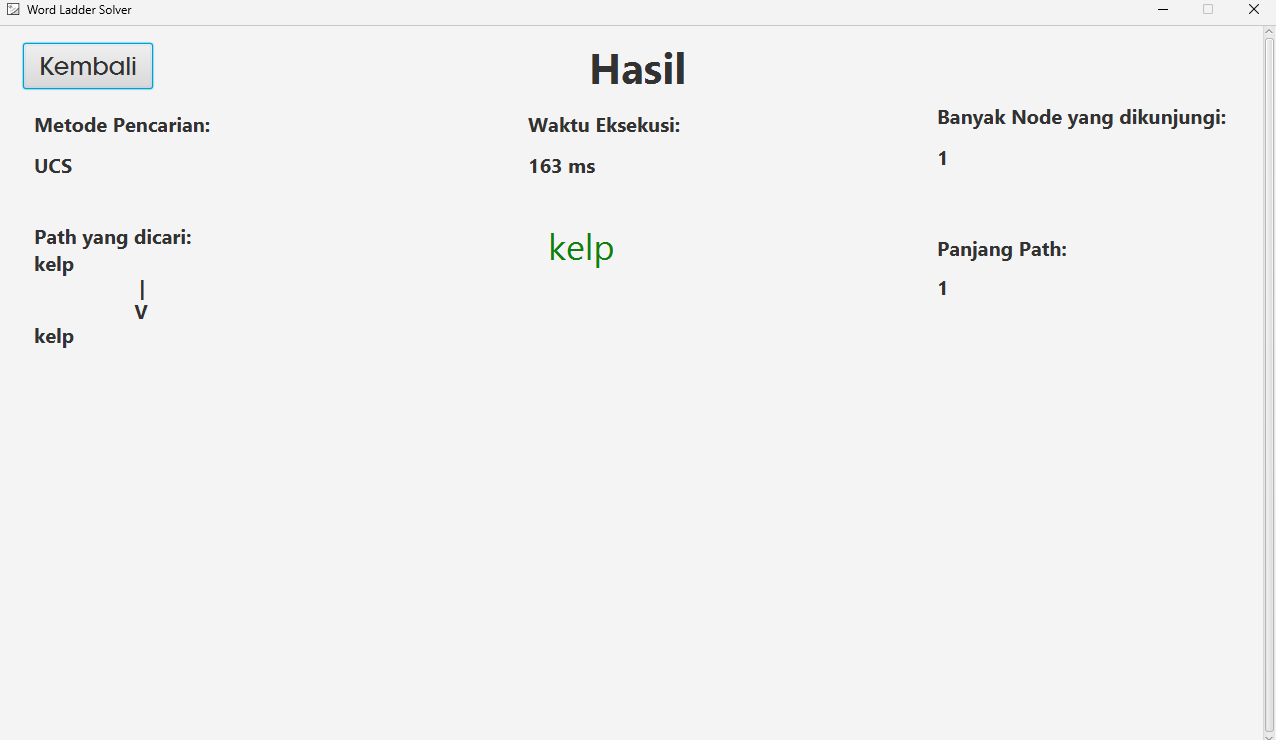
A screenshot of a computer

Description automatically generated

A white background with green text

Description automatically generated

1. KELP -> KELP
2. UCS



1. Greedy Best First Search

A screenshot of a computer

Description automatically generated

1. A\*

**A screenshot of a computer

Description automatically generated**

**Bab 6: Analisis Perbandingan Algoritma**

Berdasarkan hasil testing pada program, ketiga algoritma tersebut dapat kita analisis sebagai berikut:

1. UCS (ekuivalen dengan BFS) dan A\* berhasil menemukan path paling optimal, sedangkan GBFS belum tentu berhasil menemukan path paling optimal
2. GBFS memiliki efisiensi yang paling tinggi karena memiliki waktu eksekusi relatif paling rendah dibandingkan dengan algoritma lain, namun hasil yang diberikan oleh GBFS tidak selalu optimal.
3. A\* lebih efisien daripada UCS selain karena keduanya berhasil menemukan path paling optimal, A\* mengunjungi banyak node yang jauh lebih sedikit dibandingkan dengan UCS dan waktu yang dibutuhkan tidak jauh berbeda (atau terkadang lebih cepat sedikit dibandingkan dengan UCS).
4. Perbedaan waktu eksekusi ketiga algoritma di atas relatif tidak jauh berbeda.

Oleh karena itu, algoritma yang paling cocok untuk mencari solusi Word Ladder adalah A\* karena algoritma tersebut berhasil menemukan path terpendek dari Word Ladder dan merupakan algoritma yang paling efisien dibanding algoritma yang lain

**Bab 7: Implementasi Bonus**

Bonus yang dikerjakan adalah GUI. GUI program ini menggunakan JavaFX versi 22.0.1

Berikut adalah susunan layar GUI yang dibuat untuk tugas ini.

1. Layar Landing

A screenshot of a video game

Description automatically generated

Keterangan: Tombol “Mulai!” berfungsi untuk berpindah ke layar input utama (dan melakukan pemisahan kata dari dictionary berdasarkan panjang hurufnya)

1. Layar Input Utama

A screenshot of a computer

Description automatically generated

Keterangan:

1. Tombol “Kembali” berfungsi untuk kembali ke layar landing
2. Input teks kata awal dan kata akhir bersifat **case insensitive**.
3. Metode pencarian adalah sebuah dropdown yang berisi pilihan algoritma pencarian pada tugas kecil ini, yaitu UCS, A\*, dan Greedy Best First Search.
4. Tombol “Cari” berfungsi untuk melakukan pencarian path menggunakan algoritma yang telah dipilih.
5. Layar Hasil

A screenshot of a computer

Description automatically generated

Keterangan:

1. Tombol “Kembali” berfungsi untuk kembali ke layar input
2. Di layar ini terdapat *scrollbar* untuk melakukan scrolling solusi ke bawah jika path yang diberikan panjang.
3. Terdapat keterangan hasil pencarian yang tertera di dalam layar ini.

**Lampiran**

Link Repository: <https://github.com/DeltDev/Tucil3_13522036>

Check list program:

|  |  |  |
| --- | --- | --- |
| **Poin** | **Ya** | **Tidak** |
| 1. Program berhasil dijalankan. | V |  |
| 1. Program dapat menemukan rangkaian kata dari *start word* ke *end word* sesuai aturan permainan dengan algoritma UCS | V |  |
| 1. Solusi yang diberikan pada algoritma UCS optimal | V |  |
| 1. Program dapat menemukan rangkaian kata dari *start word* ke *end word* sesuai aturan permainan dengan algoritma *Greedy Best First Search* | V |  |
| 1. Program dapat menemukan rangkaian kata dari *start word* ke *end word* sesuai aturan permainan dengan algoritma A\* | V |  |
| 1. Solusi yang diberikan pada algoritma A\* optimal | V |  |
| 1. **[Bonus]:** Program memiliki tampilan GUI | V |  |