

LAPORAN TUGAS BESAR 2 IF2211 STRATEGI ALGORITMA

SEMESTER 2 TAHUN 2024/2025

Pemanfaatan Algoritma BFS dan DFS dalam Pencarian Recipe pada Permainan

Little Alchemy 2



Oleh:

Kelompok JumboUNCUT

13523018 - Raka Daffa Iftikhaar

13523038 - Abrar Abhirama Widyadhana

13523055 - Muhammad Timur Kanigara

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025**

DAFTAR ISI

DAFTAR ISI	2
BAB I	
DESKRIPSI TUGAS	3
BAB II	
LANDASAN TEORI	5
2.1 Graf dan Penjelajahan Graf	5
2.2 Breadth First Search (BFS)	5
2.3 Depth First Search (DFS)	6
2.4 Bidirectional Search	6
2.5 Multithreading	6
2.6 Aplikasi Web	7
BAB III	
ANALISIS PEMECAHAN MASALAH	8
3.1 Penerapan Pada BFS	8
3.1.1 Langkah - Langkah Pemecahan Masalah	8
3.1.2 Proses Pemetaan Masalah	9
3.1.3 Fitur Fungsional dan Arsitektur	10
3.1.4 Contoh Ilustrasi Kasus	11
3.2 DFS	12
3.2.1 Langkah - Langkah Pemecahan Masalah	12
3.2.2 Proses Pemetaan Masalah	12
3.2.3 Fitur Fungsional dan Arsitektur	13
3.2.4 Contoh Ilustrasi Kasus	13
3.3 BDS	14
3.3.1 Langkah - Langkah Pemecahan Masalah	14
3.3.2 Proses Pemetaan Masalah	15
3.3.3 Fitur Fungsional dan Arsitektur	16
1. Pencarian Jalur Terpendek	16
3.3.4 Contoh Ilustrasi Kasus	16
BAB IV	
IMPLEMENTASI DAN PENGUJIAN	18
4.1 Spesifikasi teknis program	18
4.1.1 Backend (Golang)	18
4.1.3 Frontend (React/Javascript)	22
4.2 Penjelasan Tata Cara Penggunaan Program	24
4.2.1 Antarmuka Awal	24
4.2.2 Antarmuka Utama	24
4.3 Hasil Pengujian	26
4.3.1 Uji BFS	26
4.3.2 Uji DFS	30

4.3.3 Uji BDS	35
4.3.4 Uji Bonus	38
4.3.5 Uji Error	40
4.4 Analisis Hasil Pengujian	40
BAB V	
KESIMPULAN, SARAN, DAN REFLEKSI	42
5.1 Kesimpulan	42
5.2 Saran	42
5.3 Refleksi	43
LAMPIRAN	44
DAFTAR PUSTAKA	45

BAB I

DESKRIPSI TUGAS

Little Alchemy 2 merupakan permainan berbasis web / aplikasi yang dikembangkan oleh Recloak yang dirilis pada tahun 2017, permainan ini bertujuan untuk membuat 720 elemen dari 4 elemen dasar yang tersedia yaitu *air*, *earth*, *fire*, dan *water*. Permainan ini merupakan sekuel dari permainan sebelumnya yakni Little Alchemy 1 yang dirilis tahun 2010.

Mekanisme dari permainan ini adalah pemain dapat menggabungkan kedua elemen dengan melakukan *drag and drop*, jika kombinasi kedua elemen valid, akan memunculkan elemen baru, jika kombinasi tidak valid maka tidak akan terjadi apa-apa. Permainan ini tersedia di *web browser*, Android atau iOS

Pada Tugas Besar pertama Strategi Algoritma ini, mahasiswa diminta untuk menyelesaikan permainan Little Alchemy 2 ini dengan menggunakan strategi Depth First Search dan Breadth First Search.

Komponen-komponen dari permainan ini antara lain:

1. Elemen dasar

Dalam permainan Little Alchemy 2, terdapat 4 elemen dasar yang tersedia yaitu *water*, *fire*, *earth*, dan *air*, 4 elemen dasar tersebut nanti akan *di-combine* menjadi elemen turunan yang berjumlah 720 elemen.



Gambar 2. Elemen dasar pada Little Alchemy 2

2. Elemen turunan

Terdapat 720 elemen turunan yang dibagi menjadi beberapa *tier* tergantung tingkat kesulitan dan banyak langkah yang harus dilakukan. Setiap elemen turunan memiliki *recipe* yang terdiri atas elemen lainnya atau elemen itu sendiri.

3. *Combine Mechanism*

Untuk mendapatkan elemen turunan pemain dapat melakukan *combine* antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

BAB II

LANDASAN TEORI

2.1 Graf dan Penjelajahan Graf

Graf adalah struktur data yang terdiri atas simpul (*node*) dan sisi (*edge*) yang menghubungkan pasangan simpul. Dalam konteks permainan *Little Alchemy 2*, setiap simpul merepresentasikan satu elemen dan sisi menunjukkan kemungkinan kombinasi dua elemen untuk membentuk elemen baru, karena setiap kombinasi memiliki arah dari bahan (*ingredients*) menuju hasil. Dalam hal ini, graf yang digunakan termasuk graf terarah.

Penjelajahan graf (*graph traversal*) adalah proses mengunjungi simpul-simpul dalam graf secara sistematis. Traversal digunakan untuk mencari jalur dari simpul awal ke simpul tujuan. Terdapat dua metode traversal yang umum digunakan dalam penjelajahan graf, yaitu BFS (*Breadth First Search*) dan DFS (*Depth First Search*). Pemilihan algoritma traversal sangat bergantung pada kebutuhan pencarian, seperti apakah ingin menemukan solusi tercepat (jumlah langkah minimum) atau mengeksplorasi semua kemungkinan jalur yang ada.

Traversal juga dapat diperluas menjadi teknik optimasi seperti *Bidirectional Search*, yaitu pencarian dilakukan secara bersamaan dari simpul awal dan simpul tujuan. Teknik ini membantu ketika ukuran graf sangat besar dan kedalaman solusi cukup panjang, karena dapat mengurangi jumlah simpul yang perlu dieksplorasi secara signifikan.

Dalam permainan *Little Alchemy 2*, traversal sangat penting untuk menyusun urutan kombinasi elemen dari bahan dasar menuju elemen target. Setiap langkah kombinasi dapat direpresentasikan sebagai perpindahan dari satu simpul ke simpul lain melalui sisi yang memiliki tanda arah. Penelusuran graf ini tidak hanya memungkinkan untuk menemukan *recipe* terpendek, tetapi juga memungkinkan pencarian seluruh jalur alternatif (*multiple recipe*) yang menghasilkan elemen yang sama, namun memanfaatkan pencarian dengan paralelisasi melalui konsep multithreading.

2.2 Breadth First Search (BFS)

Breadth-First Search (BFS) adalah algoritma penjelajahan graf yang mengeksplorasi semua simpul pada level yang sama terlebih dahulu sebelum berpindah ke level berikutnya. BFS menggunakan struktur data antrian (*queue*) dan cocok digunakan untuk menemukan jalur terpendek (*shortest path*) dalam graf tak berberat.

Dalam implementasi aplikasi *Little Alchemy 2*, BFS digunakan untuk menemukan langkah-langkah kombinasi elemen secara menyeluruh dari elemen dasar ke elemen target. Karena BFS menjelajahi graf secara menyeluruh dan sistematis, ia efektif dalam mencari

solusi optimal dalam hal jumlah langkah kombinasi yang diperlukan. Selain itu, dalam mode pencarian multiple path, BFS mampu menemukan berbagai jalur unik pembentukan elemen dengan menghindari pengulangan jalur sebelumnya.

2.3 Depth First Search (DFS)

Depth-First Search (DFS) adalah algoritma penjelajahan graf yang menjelajahi satu cabang pohon graf sedalam mungkin sebelum kembali (*backtrack*) ke simpul sebelumnya dan menjelajahi cabang lainnya. DFS menggunakan struktur data stack (baik eksplisit maupun rekursif).

DFS memiliki keunggulan dalam menemukan solusi lebih cepat dalam beberapa kasus, terutama bila solusi berada di kedalaman tertentu dan tidak terlalu banyak cabang yang harus dijelajahi. Dalam konteks permainan ini, DFS digunakan untuk mencari satu atau beberapa jalur pembentukan elemen, namun tidak menjamin bahwa jalur tersebut merupakan jalur terpendek. Oleh karena itu, DFS umumnya lebih cocok digunakan untuk mode pencarian *multiple path*, di mana eksplorasi jalur alternatif lebih diutamakan dibanding optimalitas panjang jalur.

2.4 Bidirectional Search

Bidirectional Search adalah teknik pencarian yang menjalankan dua pencarian sekaligus, yaitu satu dari simpul awal (elemen dasar) dan satu lagi dari simpul tujuan (elemen target), hingga keduanya akan bertemu di tengah. Dengan metode ini, waktu pencarian dapat dikurangi secara signifikan dibandingkan pencarian satu arah penuh.

Namun, implementasi *bidirectional search* dalam permainan *Little Alchemy 2* ini lebih menantang, karena arah pembentukan elemen hanya bisa dari bahan ke hasil (graf terarah). Oleh karena itu, perlu dilakukan modifikasi seperti membalik sebagian relasi atau menggunakan struktur data tambahan untuk menyimpan invers kombinasi secara eksplisit. Jika berhasil diterapkan, *bidirectional search* ini dapat menjadi strategi tambahan yang mempercepat pencarian baik dalam mode *shortest path* atau *multiple path*.

2.5 Multithreading

Multithreading adalah teknik pemrograman yang memungkinkan program untuk menjalankan beberapa bagian secara paralel dalam satu waktu. Ini dapat meningkatkan performa program, terutama pada proses-proses yang bersifat independen dan dapat dijalankan bersamaan.

Dalam aplikasi ini, multithreading digunakan untuk mempercepat proses pencarian jalur kombinasi, terutama ketika mode pencarian “semua solusi” diaktifkan. Setiap jalur DFS dapat dijalankan dalam thread terpisah untuk memanfaatkan core prosesor secara maksimal.

Dengan demikian, waktu eksekusi pencarian yang biasanya memakan waktu lama dapat dipersingkat secara signifikan.

2.6 Aplikasi Web

Aplikasi web yang dibangun merupakan alat bantu visualisasi dan pencarian kombinasi elemen dalam permainan *Little Alchemy 2*. Aplikasi ini dirancang untuk menerima masukan berupa element target dari pengguna, kemudian menampilkan jalur kombinasi dari elemen dasar menggunakan algoritma BFS, DFS, atau BDS.

Aplikasi ini terdiri atas dua bagian utama, yaitu:

- Frontend : Dibuat menggunakan [React.js](#) , yang mana menyediakan antarmuka interaktif agar pengguna dapat memilih elemen, metode pencarian, hingga melihat hasil kombinasi dalam bentuk jalur atau pohon.
- Backend : Dibuat menggunakan bahasa Golang untuk melakukan proses pencarian berdasarkan data kombinasi yang tersedia. Algoritma BFS dan DFS diimplementasikan di bagian ini agar proses pencarian tetap efisien dan direspon dengan cepat bahkan untuk elemen yang kompleks.

Aplikasi web ini mendukung dua mode pencarian, yaitu “*shortest path*” untuk menemukan kombinasi paling pendek dan juga “*multiple path*” untuk menentukan berbagai kombinasi yang tersedia. Selain itu, backend mendukung pemrosesan paralel menggunakan konsep *multithreading* agar waktu respons tetap cepat, meskipun pencarian dilakukan pada graf yang besar/luas. Visualisasi *recipe* ditampilkan dalam bentuk *tree* yang menjelaskan tahapan kombinasi dari elemen dasar hingga elemen target. Selain itu, visualisasi pencarian pada aplikasi dapat ditampilkan secara *live* saat pencarian berlangsung.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Penerapan Pada BFS

3.1.1 Langkah - Langkah Pemecahan Masalah

1. Inisialisasi

Langkah awal dimulai dengan memuat dan memproses file JSON yang berisi daftar elemen dan resepnya. Elemen dasar seperti Air, Earth, Fire, dan Water langsung ditandai sebagai tersedia.

2. Pemrosesan Data

Pada fungsi untuk mengurus single path, program akan membangun sebuah queue. Berdasar pada queue itu, program akan memproses elemen yang paling depan menggunakan metode FIFO, seperti Air, Earth, Fire, dan Water. Pada fungsi untuk mengurus multiple paths, program akan menghitung terlebih dahulu berapa banyak kombinasi yang mungkin berdasar pada file JSON. Setelah itu, akan dipanggil fungsi FindPathBFS() untuk mencari jalur tercepat atau jalur pertama. Jika kombinasi lebih dari satu, maka akan dipanggil worker untuk mencari jalur ke target dengan kombinasi-kombinasi yang berbeda. Proses pada setiap worker sama dengan fungsi FindPathBFS().

3. Pembangunan Jalur

Pembangunan jalur dari program ini dibagi menjadi dua seperti pemrosesan data. Pertama untuk fungsi FindPathBFS(), setelah elemen target ditemukan, program akan membangun jalur resep dari base elemen ke target menggunakan fungsi buildRecipePath(). Sedangkan untuk FindMultiplePathsBFS() akan ada beberapa jalur yang dibuat menggunakan buildDiversePath(). Jalur pertama akan dibuat dengan fungsi buildRecipePath() sama seperti single BFS sedangkan jalur lainnya akan menggunakan worker untuk membuatnya dan menggunakan metode yang sama dengan buildRecipePath(). Jalur-jalur itu akan disimpan pada allFoundPaths jika memang unik.

4. Pengelolaan Paralelisme dan Jalur Alternatif

Paralelisme atau multithreading dan jalur alternatif hanya kami implementasikan pada fungsi FindMultiplePathsBFS(). Pada fungsi tersebut, paralelisme dijalankan dengan menggunakan banyak worker. Setiap kombinasi unik akan dijalankan tiga worker sehingga pencarian tidak satu per satu. Worker dijalankan dengan go func() dan dikelola dengan sync.WaitGroup. Terdapat worker tambahan juga untuk melakukan pencarian lebih bebas jika semua path masih ada yang belum ditemukan. Pada jalur alternatif, setiap worker akan mencoba untuk mencari jalur

berbeda ke target menggunakan buildDiversePath(). Hasil jalur yang unik akan ditambahkan ke allFoundPaths().

5. Penanganan Duplikasi dan Loop

Setiap jalur yang ditemukan akan diidentifikasi dengan dua cara yaitu comboKey untuk memastikan setiap kombinasi dihitung sekali dan pathID untuk memastikan urutan langkah resep yang sama tidak dimasukkan dua kali. Untuk penanganan loop, setiap worker menggunakan map seperti localVisited, discovered, dan parent untuk menandai elemen yang sudah dikunjungi atau sudah ditemukan. Selain itu, pada setiap pasangan elemen juga dicatat di localVisited agar kombinasi yang sama tidak diproses berulang kali.

6. Penyajian dan Ekspor Hasil

Pada fungsi FindPathBFS(), penyajian dan eksplor hasil dilakukan dengan membangun urutan langkah resep dari base element ke target menggunakan buildRecipePath(). Hasil jalur tersebut akan disimpan pada cache untuk memudahkan pencarian selanjutnya. Untuk fungsi FindMultiplePathsBFS(), penyajian dan eksplorasi hasil dilakukan dengan mengunci mutex yang menandakan semua worker sudah selesai bekerja. Setelah itu, seluruh isi allFoundPaths akan disalin kemudian akan ditampilkan.

3.1.2 Proses Pemetaan Masalah

Proses pemetaan masalah pada program kami (khususnya pada fungsi-fungsi BFS) dilakukan dengan cara mengubah masalah pencarian jalur pembuatan elemen alkimia menjadi masalah pencarian jalur pada graf.

1. Representasi Masalah

- Node

Setiap elemen (misal: Air, Api, Steam, dsb) direpresentasikan sebagai node pada graf.

- Edge

Setiap resep (kombinasi dua elemen menghasilkan elemen baru) direpresentasikan sebagai edge atau transisi dari dua node (bahan) ke satu node (hasil).

- Base Elements

Elemen dasar (Air, Earth, Fire, Water) adalah node awal yang tidak bisa dibentuk dari elemen lain.

2. Pemetaan ke Algoritma BFS

Pemetaan ini bertujuan untuk mencari urutan langkah (jalur) dari base elements ke target element dengan memanfaatkan resep-resep yang tersedia. Terdapat

beberapa komponen dalam pemetaan ini. Pertama, queue yang digunakan untuk menyimpan elemen yang akan diproses berikutnya, sesuai dengan prinsip BFS. Kedua adalah visited/discovered yang berupa tipe data map. Ini digunakan untuk memindai elemen yang sudah dikunjungi agar tidak terjadi loop.

3. Pemetaan Jalur

Pemetaan jalur dibagi menjadi dua yaitu parent map dan build path. Parent map digunakan untuk mencatat resep yang membentuknya setiap ada elemen baru ditemukan. Build path digunakan untuk menyimpan path penelusuran balik dari target untuk membangun urutan langkah resep.

4. Pemetaan Jalur Alternatif

Pada pemetaan jalur, untuk mencari lebih dari satu jalur unik, digunakan worker paralel dengan strategi pencarian berbeda dan kombinasi bahan berbeda. Jalur dianggap unik jika kombinasi bahan atau urutan langkahnya berbeda.

5. Pemetaan Hasil

Hasil pencarian berupa urutan resep ([]Recipe) yang membentuk target element, dikembalikan ke pemanggil untuk ditampilkan atau diproses lebih lanjut.

3.1.3 Fitur Fungsional dan Arsitektur

Pada program BFS kami, terdapat beberapa fitur fungsional seperti

1. Pencarian Jalur untuk Elemen

Mencari urutan resep dari base elemen ke target elemen menggunakan algoritma BFS dan menghasilkan suatu solusi.

2. Pencarian Banyak Jalur Unik

Mencari beberapa jalur unik menuju target elemen dan mendukung pencarian paralel untuk efisiensi dan variasi solusi.

3. Pembangunan Jalur (Path Building)

Menyusun urutan langkah resep dari base elements ke target berdasarkan parent map hasil BFS.

4. Manajemen Kombinasi Resep

Mengidentifikasi semua kombinasi bahan unik yang bisa menghasilkan target element.

5. Pencegahan Duplikasi & Siklus

Menggunakan map untuk menandai kombinasi yang sudah ditemukan dan mencegah loop pada pencarian.

6. Pengelolaan Cache

Menyediakan cache untuk hasil pencarian agar pencarian berikutnya lebih cepat.

7. Sorting & Strategi Pencarian

Mendukung berbagai strategi urutan pencarian untuk memperkaya variasi solusi.

Sedangkan untuk arsitektur, program kami memiliki arsitektur seperti berikut.

1. Graph-Based Search

Masalah dipetakan ke graf dengan aturan node adalah elemen dan edge adalah resep. BFS digunakan sebagai algoritma utama pencarian jalur.

2. Concurrency/Parallelism

Menggunakan goroutine dan sync primitives (WaitGroup, Mutex, Atomic) untuk pencarian paralel. Channel (pathChan) digunakan untuk komunikasi hasil antar worker.

3. Modular Function Design

Fungsi-fungsi dipisah sesuai tanggung jawab seperti pencarian jalur, pembangunan jalur, manajemen kombinasi, sorting, dan lain-lain.

4. Data Structure

Map digunakan untuk cache, visited, parent, discovered, dan manajemen kombinasi. List/queue dari package container/list untuk antrian BFS.

5. Error Handling & Logging

Error dikembalikan ke pemanggil. Proses dan hasil penting dicetak ke terminal/log untuk debugging dan monitoring.

3.1.4 Contoh Ilustrasi Kasus

Misalkan kita ingin mencari cara untuk membuat Geyser. Pertama, menggunakan BFS Single Path.

1. Titik Awal: Memiliki empat elemen dasar.

2. Target Elemen: Elemen “Geyser”.

3. Proses BFS

- a. BFS akan memproses elemen dasar terlebih dahulu yaitu Air, Water, Fire, dan Earth.

- b. Keempat elemen tersebut diproses satu per satu melalui queue.

- c. Kombinasi elemen yang didapatkan dari keempat elemen tersebut dimasukkan ke dalam queue.

- d. Sebagai contoh, Air + Fire -> Steam. Steam akan dimasukkan ke queue.

- e. Iterasi terus dijalankan sampai mendapatkan target yaitu Geyser.

- f. Iterasi berhenti ketika Geyser didapatkan dari Steam + Earth -> Geyser

- g. Untuk multipath, prosesnya sama namun sebelum mulai akan dihitung dulu total kombinasi yang bisa kemudian baru dilakukan prosesnya.

- h. Setelah itu akan diakhiri pencocokan path, bila ada path yang berbeda akan dimasukkan, bila sama tidak akan dipakai.

4. Hasil jalur Geyser ditemukan dengan dua cara yaitu:

- A

- B

3.2 DFS

3.2.1 Langkah - Langkah Pemecahan Masalah

1. Inisialisasi

Menyiapkan struktur data untuk elemen dasar (Air, Earth, Water, Fire) dan menyiapkan cache untuk menyimpan elemen yang sudah diketahui dapat dibuat.

2. Fungsi Pemeriksaan Kemampuan Pembuatan

Melalui fungsi `isCreatable()`, algoritma memeriksa apakah suatu elemen dapat dibuat dari elemen-elemen yang ada, dengan mempertimbangkan

- a. Elemen dasar selalu bisa dibuat
- b. Elemen yang sudah diketahui statusnya (dari cache)
- c. Deteksi loop untuk mencegah rekursi yang tak terbatas

3. Pembangunan Jalur

Menggunakan fungsi `buildOrderedPath()`, untuk mencari jalur optimal dari elemen dasar ke elemen target dengan pendekatan bottom-up, yaitu:

- a. Mencari resep yang membuat elemen target
- b. Memastikan bahan-bahan dalam resep dapat dibuat
- c. Membangun jalur secara rekursif, menyelesaikan prasyarat yang ada terlebih dahulu.

4. Penanganan Duplikasi

Menggunakan fungsi `removeDuplicateRecipes()` untuk menghilangkan resep yang berulang sehingga dapat mengoptimalkan jalur.

5. Pencarian Jalur Alternatif

Dengan menggunakan `findMultiplePathsDFS()`, algoritma mencari jalur-jalur alternatif dengan multi-threading menggunakan goroutines untuk bisa mengeksplorasi secara paralel.

6. Validasi Jalur

Memastikan setiap jalur valid dengan memeriksa ketersediaan bahan-bahan pada setiap langkah.

3.2.2 Proses Pemetaan Masalah

Masalah pembuatan elemen dalam program ini dapat dipetakan ke dalam struktur graf berarah sebagai berikut:

1. Representasi Graf

- a. **Node:** Elemen-elemen dalam permainan (Air, Earth, Water, Fire, dan elemen turunan)
- b. **Edge:** Resep yang menghubungkan dua elemen bahan ke elemen hasil
- c. **Edge Weight:** Implisit 1 (setiap langkah pembuatan dianggap sama)

2. Karakteristik Graf

- a. **Cyclical**: Dalam permainan aslinya, resep / graf dapat mengandung siklus karena elemen A dapat digunakan untuk membuat B, elemen B untuk membuat elemen C, dan elemen C digunakan untuk membuat elemen A. Akan tetapi, dalam program ini sudah dibatasi bahwa elemen target hanya bisa dibuat oleh tier elemen yang lebih rendah, sehingga jika menggunakan contoh di atas, resep untuk membuat elemen A menggunakan elemen C sudah dihapus.
- b. **Directed**: Resep hanya berjalan satu arah (bahan → hasil)
- c. **Multigraf**: Beberapa resep berbeda dapat menghasilkan elemen yang sama.

3. Pemetaan Algoritma DFS

DFS digunakan untuk melakukan eksplorasi mendalam pada graf. Algoritma akan mencari jalur dari elemen dasar (root) ke elemen target. Pencarian dilakukan secara bottom-up untuk memastikan bahwa semua prasyarat terpenuhi.

3.2.3 Fitur Fungsional dan Arsitektur

Implementasi DFS menyediakan beberapa fitur fungsional:

1. **Pencarian Jalur Tunggal**: Menemukan satu jalur optimal untuk membuat elemen target.
2. **Pencarian Multipath**: Menemukan beberapa jalur alternatif untuk mencapai elemen target.
3. **Optimasi Efisiensi**: Memanfaatkan caching untuk menghindari perhitungan berulang.
4. **Deteksi Ketersediaan**: Memverifikasi apakah elemen dapat dibuat dari elemen dasar.
5. **Pemrosesan Paralel**: Untuk *multipath*, pencarian dilakukan secara paralel untuk meningkatkan performa.

Arsitektur DFS dibangun menggunakan komponen-komponen berikut:

1. **Cache Elemen**: Menyimpan informasi tentang elemen yang dapat dibuat
2. **Cache Jalur**: Menyimpan jalur yang sudah ditemukan untuk elemen tertentu.
3. **Sistem Verifikasi**: Memastikan jalur yang ditemukan valid dengan semua bahannya tersedia di setiap langkah.
4. **Sistem Pengurutan**: Memprioritaskan resep berdasarkan ketersediaan bahan dan penggunaan elemen dasar.

3.2.4 Contoh Ilustrasi Kasus

Misalkan kita ingin mencari jalur untuk membuat elemen “Brick” menggunakan DFS. Ilustrasinya:

1. Titik Awal: Memiliki empat elemen dasar.
2. Target: Elemen “Brick”

3. Proses DFS
 - a. DFS menentukan bahwa “Brick” dapat dibuat dari “Mud” dan “Fire”
 - b. DFS menyelam untuk mencari jalur membuat “Mud”
 - c. DFS menemukan bahwa “Mud” dapat dibuat dari “Earth” dan “Water”
 - d. Keduanya adalah elemen dasar, sehingga jalur ke “Mud” ditemukan
 - e. DFS kembali ke “Brick” dan mencatat bahwa “Fire” sudah ada karena elemen dasar
 - f. DFS menyusun jalur lengkap: Earth + Water → Mud, lalu Mud + Fire → Brick
4. Hasil: Jalur pembuatan “Brick” ditemukan dengan urutan:
 - a. Langkah 1 : Earth + Water → Mud
 - b. Langkah 2 : Mud + Fire → Brick
5. Jalur Alternatif: Jika diminta, DFS dapat menemukan jalur alternatif, misalnya jika Brick memiliki resep lain

Pendekatan DFS ini memberikan keuntungan dalam menemukan jalur yang optimal dengan cepat karena fokus pada eksplorasi kedalaman dahulu, yang sesuai dengan sifat alami dari proses pembuatan elemen dimana elemen dasar terus dikombinasikan menjadi elemen yang lebih kompleks.

3.3 BDS

3.3.1 Langkah - Langkah Pemecahan Masalah

1. Inisialisasi

Menyiapkan dua antrian: queueForward untuk pencarian maju dan queueBackward untuk pencarian mundur. Menyiapkan dua set elemen yang sudah dikunjungi: visitedForward dan visitedBackward untuk melacak elemen yang sudah dieksplorasi dari masing-masing arah. Menyiapkan dua map parent: parentForward untuk merekonstruksi jalur maju (menyimpan resep yang menghasilkan setiap elemen dari arah maju) dan parentBackward untuk merekonstruksi jalur mundur (menyimpan resep yang menghasilkan setiap elemen dari arah mundur). Antrian maju diinisialisasi dengan elemen-elemen dasar (Air, Earth, Fire, Water). Antrian mundur diinisialisasi dengan elemen target yang dicari.

2. Proses Eksplorasi Bergantian

Algoritma secara bergantian melakukan ekspansi satu level dari antrian maju dan satu level dari antrian mundur.

- Eksplorasi Maju: Mengambil elemen dari queueForward. Untuk setiap elemen yang diambil, algoritma mencari resep-resep yang menggunakan

elemen tersebut sebagai salah satu bahan. Jika bahan pasangannya sudah dikunjungi dari arah maju (visitedForward), elemen hasil resep ditambahkan ke queueForward dan visitedForward, serta resep tersebut dicatat di parentForward sebagai cara untuk menghasilkan elemen hasil.

- Eksplorasi Mundur: Mengambil elemen dari queueBackward (elemen ini adalah hasil yang sedang dicari dari arah mundur). Algoritma mencari semua resep yang menghasilkan elemen tersebut. Bahan-bahan dari resep yang ditemukan ditambahkan ke queueBackward dan visitedBackward, serta resep tersebut dicatat di parentBackward sebagai cara untuk menghasilkan elemen yang sedang diproses (yaitu, elemen yang diambil dari antrian mundur).

3. Deteksi Penemuan

Di setiap langkah eksplorasi (baik maju maupun mundur), algoritma memeriksa apakah elemen yang baru saja dikunjungi sudah ada di set visited dari arah yang berlawanan. Jika elemen yang dieksplorasi dari arah maju (currF) ditemukan di visitedBackward, atau elemen yang dieksplorasi dari arah mundur (currB_as_RESULT atau bahan-bahannya) ditemukan di visitedForward, maka titik pertemuan (meetingNode) telah ditemukan dan pencarian dihentikan.

4. Rekonstruksi Jalur

Setelah titik pertemuan (meetingNode) ditemukan, jalur lengkap dari elemen dasar ke elemen target direkonstruksi. Jalur ini dirangkai dengan menggabungkan dua segmen, Segmen dari elemen dasar ke meetingNode, direkonstruksi menggunakan parentForward. Kemudian Segmen dari meetingNode ke elemen target, direkonstruksi menggunakan parentBackward (kemudian urutannya dibalik). Kedua segmen jalur tersebut kemudian digabungkan untuk membentuk jalur resep lengkap dari elemen dasar hingga elemen target.\

5. Pencarian Jalur ganda (Multiplepaths)

Untuk mencari beberapa jalur berbeda, program meluncurkan beberapa proses pencarian BDS secara paralel menggunakan goroutine. Setiap goroutine menjalankan pencarian BDS terpisah (FindPathBDS) untuk elemen target yang sama. Hasil jalur dari setiap goroutine dikumpulkan. Duplikasi jalur diidentifikasi dan dieliminasi menggunakan identifier jalur unik. Proses pengumpulan jalur berlanjut hingga jumlah jalur unik yang diminta (maxRecipes) terpenuhi atau semua goroutine selesai.

3.3.2 Proses Pemetaan Masalah

1. Representasi Graf

Node merepresentasikan elemen, dan edge merepresentasikan resep (kombinasi dua bahan menghasilkan satu hasil).

2. Karakteristik Graf

Graf bersifat directed (resep satu arah) dan dapat dianggap sebagai multigraph (banyak resep menghasilkan elemen yang sama). Pembatasan pada resep (menggunakan elemen tier lebih rendah) membantu mengurangi atau menghilangkan siklus yang relevan untuk pencarian reachable paths.

3. Pemetaan Algoritma BDS

BDS sangat cocok untuk mencari jalur terpendek dalam graf ini. Pencarian maju mengeksplorasi ruang solusi dari elemen yang dapat dibuat, sementara pencarian mundur mengeksplorasi ruang solusi dari elemen prasyarat yang dibutuhkan untuk target. Pertemuan kedua pencarian menjamin ditemukannya jalur terpendek (seperti BFS), tetapi dengan potensi eksplorasi ruang yang lebih sedikit dibandingkan BFS tunggal pada graf yang luas.

3.3.3 Fitur Fungsional dan Arsitektur

1. Pencarian Jalur Terpendek

Menemukan satu jalur resep terpendek yang memungkinkan pembuatan elemen target.

2. Pencarian Multipath

Menemukan beberapa jalur resep berbeda untuk mencapai elemen target dengan memanfaatkan paralelisme.

3. Efisiensi

Berpotensi menemukan jalur terpendek lebih cepat dibandingkan BFS tunggal pada graf dengan faktor percabangan tinggi.

4. Deteksi Ketersediaan

Secara implisit memverifikasi apakah elemen target dapat dibuat dari elemen dasar jika jalur ditemukan.

3.3.4 Contoh Ilustrasi Kasus

Misalkan kita ingin mencari jalur untuk membuat elemen “Brick” menggunakan BDS.

1. Titik Awal: Pencarian Maju dimulai dari elemen dasar {Air, Earth, Fire, Water}. Pencarian Mundur dimulai dari elemen target {Brick}.
2. Proses BDS:
 - Iterasi 1 (Maju): Ambil elemen dasar (misal: Earth). Kombinasikan dengan elemen dasar lain yang sudah dikunjungi (Air). Hasil: Dust. Dust

ditambahkan ke antrian maju dan visitedForward. parentForward[Dust] = {Earth, Air => Dust}. Antrian maju kini berisi elemen dasar + elemen tier 1.

- Iterasi 1 (Mundur): Ambil target (Brick). Cari resep yang menghasilkan Brick (misal: Mud + Fire => Brick, Clay + Stone => Brick). Bahan-bahan Mud, Fire, Clay, Stone ditambahkan ke antrian mundur dan visitedBackward. parentBackward[Brick] = {Mud, Fire => Brick} (atau salah satu resep yang ditemukan). Cek apakah Mud, Fire, Clay, Stone ada di visitedForward. Fire ditemukan di visitedForward. Pertemuan terjadi di "Fire".

3. Rekonstruksi Jalur:

- Segmen Maju (Fire ke Dasar): Fire adalah elemen dasar. Rekonstruksi dari parentForward untuk Fire menghasilkan jalur kosong [].
- Segmen Mundur (Brick ke Fire): Rekonstruksi dari parentBackward untuk Brick. parentBackward[Brick] adalah resep yang menggunakan sebagai hasil (misal: Mud + Fire => Brick). Mundur ke bahan-bahan (Mud, Fire). Karena kita menuju Fire (titik pertemuan), kita pilih Fire. Rekonstruksi berhenti di Fire. Segment mundur (sebelum dibalik): [{Mud, Fire => Brick}]. Dibalik menjadi: [{Mud, Fire => Brick}].
- Jalur Lengkap: Menggabungkan segmen maju dan mundur: [] + [{Mud, Fire => Brick}] = [{Mud, Fire => Brick}].

4. Hasil: Jalur pembuatan "Brick" ditemukan dengan urutan: Langkah 1 : Mud + Fire → Brick

Jika BDS melanjutkan eksplorasi (misalnya, jika ada goroutine lain atau jika resep lain dieksplorasi lebih dulu di salah satu arah), ia bisa menemukan titik pertemuan lain (misalnya, melalui resep Clay + Stone => Brick) dan merekonstruksi jalur yang berbeda.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi teknis program

4.1.1 Backend (Golang)

Backend diimplementasikan menggunakan bahasa pemrograman Golang, bertanggung jawab atas pengelolaan data resep, implementasi algoritma pencarian graf, dan penyediaan API untuk frontend.

A. Struktur data utama:

- **Struct Recipe**

Berfungsi untuk mempresentasikan satu kombinasi resep dalam permainan. Memiliki 3 struktur yaitu Result (string, nama elemen hasil), Ingredient1 (string, nama bahan pertama), dan Ingredient2 (string, nama bahan kedua). Struktur ini digunakan untuk menyimpan, memproses, dan mengembalikan data resep.

```
type Recipe struct {
    Result    string `json:"result"`
    Ingredient1 string `json:"ingredient1"`
    Ingredient2 string `json:"ingredient2"`
}
```

- **Struct ElementImage**

Merepresentasikan pemetaan nama elemen dengan URL gambar ikonnya. Memiliki dua field: Name (string) dan ImageURL (string).

```
type ElementImage struct {
    Name    string `json:"name"`
    ImageURL string `json:"imageURL"`
}
```

- **map[string][]Recipe alchemyGraph**

Merepresentasikan graf resep dalam bentuk adjacency list. Kunci dari map ini adalah nama bahan elemen, dan nilainya adalah slice dari semua resep ([]Recipe) di mana bahan tersebut digunakan. Struktur ini dioptimalkan untuk

pencarian ke depan, menemukan resep yang bisa dibuat dari elemen yang tersedia

- **map[string]string imageMap**

Menyimpan pemetaan nama elemen ke URL gambar ikon mereka untuk akses cepat.

- **map[string][]Recipe recipeMap**

Merepresentasikan resep yang dikelompokkan berdasarkan hasil elemen. Kunci dari map ini adalah nama hasil elemen, dan nilainya adalah slice dari semua resep ([]Recipe) yang menghasilkan elemen tersebut. Struktur ini berguna untuk pencarian mundur atau untuk dengan cepat mengetahui semua cara membuat elemen tertentu.

- **map[string]bool allElementNames**

Set yang berisi semua nama elemen unik yang diketahui (elemen dasar, hasil resep, dan bahan resep), digunakan terutama untuk validasi.

- **map[string]bool allElementNames**

Set yang berisi semua nama elemen unik yang diketahui (elemen dasar, hasil resep, dan bahan resep), digunakan terutama untuk validasi.

- **Cache (map[string][]Recipe bfsPathCach, map[string][]Recipe pathCache, dll.)**

Digunakan dalam implementasi algoritma pencarian (khususnya BFS dan DFS) untuk menyimpan hasil pencarian sebelumnya, mempercepat pencarian ulang untuk elemen yang sama.

B. Fungsi dan Prosedur Utama

- **main() (main.go)**

Fungsi entry point. Menginisialisasi aplikasi dengan memanggil fungsi-fungsi pemuatan data (InitData), pembangunan graf (BuildGraph), dan pengaturan server HTTP (http.ListenAndServe). Menangani command-line flag -scrapeonly.

- **RunScraping() (scraping.go)**

Mengimplementasikan logika web scraping menggunakan library goquery untuk mengambil data resep dan URL gambar dari website target. Data mentah disimpan ke file JSON.

- **runFilter() (filter.go)**

Memuat data resep mentah, memprosesnya untuk memfilter resep yang tidak reachable dari elemen dasar dan memvalidasi logika tier resep. Data resep yang sudah difilter disimpan ke file JSON final.

- **loadRecipes(), loadImages() (data.go)**

Fungsi internal untuk membaca data resep dan gambar dari file JSON.

- **processDataToMaps() (data.go)**

Memproses data yang dibaca dari file JSON ke dalam struktur map (recipeMap, imageMap, allElementNames) di memori.

- **InitData() (data.go)**

Fungsi yang dipanggil dari main() untuk menginisialisasi semua data aplikasi menggunakan sync.Once untuk memastikan hanya dieksekusi sekali.

- **BuildGraph() (graph.go)**

Membangun representasi alchemyGraph dari recipeMap yang sudah dimuat. Menggunakan sync.Once.

- **Algoritma Pencarian (bfs.go, dfs.go, bds.go)**

- **FindPathBFS(targetElement string)**

Mencari jalur terpendek tunggal menggunakan algoritma BFS.

- **FindMultiplePathsBFS(targetElement string, maxRecipes int)**

Mencari sejumlah (maxRecipes) jalur resep berbeda menggunakan BFS dengan dukungan multithreading (goroutine).

- **FindPathDFS(targetElement string)**

Mencari satu jalur resep menggunakan algoritma DFS (tidak dijamin terpendek).

- **FindMultiplePathsDFS(targetElement string, maxRecipes int)**

Mencari sejumlah (maxRecipes) jalur resep berbeda menggunakan DFS dengan dukungan multithreading (goroutine).

- **FindPathBDS(targetElement string)**

Implementasi algoritma Bidirectional Search untuk mencari jalur terpendek. (Berdasarkan kode, mungkin masih dalam pengembangan).

- **FindMultiplePathsBDS(targetElement string, maxRecipes int)**

Mencari sejumlah (maxRecipes) jalur resep berbeda menggunakan Bidirectional Search dengan dukungan multithreading. (Berdasarkan kode, mungkin masih dalam pengembangan).

C. Fungsi Bantu Algoritma

Fungsi-fungsi seperti `getRecipes`, `isBaseElement`, `buildRecipePath`, `generatePathIdentifier`, `getPairKey`, serta fungsi-fungsi terkait multithreading dan caching yang mendukung implementasi algoritma pencarian.

- **searchHandler(w http.ResponseWriter, r *http.Request) (handlers.go)**

HTTP handler untuk endpoint /api/search. Membaca parameter permintaan dari frontend (target, algoritma, mode, max recipes), memvalidasi input, memanggil fungsi algoritma pencarian yang sesuai, mengukur waktu eksekusi, dan mengembalikan hasilnya dalam format JSON ke frontend. Menangani CORS.

- **imageHandler(w http.ResponseWriter, r *http.Request) (handlers.go)**

HTTP handler untuk endpoint /api/image. Berfungsi sebagai proxy untuk mengambil gambar elemen dari URL aslinya (dari `imageMap`) dan mengirimkannya kembali ke frontend. Ini membantu menghindari masalah CORS di sisi frontend. Menangani CORS.

D. Penggunaan Konkurenси:

- **sync.Once**

Digunakan pada `InitData` dan `BuildGraph` untuk memastikan proses inisialisasi yang mahal hanya dijalankan satu kali selama siklus hidup aplikasi.

- **sync.Mutex dan sync.RWMutex**

Digunakan untuk melindungi akses ke shared resources (seperti cache jalur dan daftar jalur yang ditemukan) dari race condition saat diakses oleh beberapa goroutine secara bersamaan, terutama pada implementasi pencarian multiple paths.

- **sync.WaitGroup**

Digunakan untuk menunggu selesainya eksekusi sekelompok goroutine, misalnya saat mencari multiple paths secara paralel.

- **atomic package**

Digunakan untuk operasi atomik (seperti menghitung nodesVisitedCount secara aman dari beberapa goroutine), menghindari kebutuhan Mutex untuk operasi sederhana.

- **Channels**

Digunakan (terutama di implementasi BDS multiple, meskipun mungkin belum sepenuhnya berfungsi) untuk komunikasi dan sinkronisasi antar goroutine, serta untuk mengumpulkan hasil.

4.1.3 Frontend (React/Javascript)

A. Struktur data utama:

- **State Components (useState)**

Berbagai state lokal dalam komponen React (App.jsx, SearchForm.jsx, SearchResults.jsx) untuk mengelola status UI, input pengguna, data hasil pencarian, status loading, error, serta state spesifik untuk animasi dan fitur live update. Contoh: appState, searchResultsData, isLoading, liveUpdateStates.

- **Struktur Data Hasil Pencarian (dari backend)**

Objek JSON yang diterima dari endpoint /api/search. Strukturnya adalah MultiSearchResponse yang mencakup searchTarget, algorithm, mode, maxRecipes (jika mode multiple), pathFound (boolean), path ([]Recipe, jika mode shortest), paths ([][]Recipe, jika mode multiple), imageURLs (map[string]string), nodesVisited (int), durationMillis (int64), dan error (string).

- **Struktur Data Tree (react-d3-tree)**

Objek atau array objek yang diformat sesuai dengan kebutuhan library react-d3-tree untuk menampilkan visualisasi graf. Setiap node memiliki properti seperti name, attributes, dan children. Struktur ini dibangun di SearchResults.jsx berdasarkan data resep dari backend.

B. Fungsi dan Prosedur Utama:

- Komponen Fungsional (React Components):

- **App()** (App.jsx)

Komponen root. Mengelola state utama aplikasi dan alur pencarian. Merender komponen SearchForm dan SearchResults secara kondisional. Menangani transisi UI dan animasi awal.

- **SearchForm({ onSearchSubmit, isLoading })** (SearchForm.jsx)

Komponen form input. Mengelola state input (target, algo, mode, maxRecipes). Menyediakan UI untuk memilih opsi pencarian. Memanggil onSearchSubmit saat form disubmit.

- **SearchResults({ results, isLoading, error }) (SearchResults.jsx)**
Komponen tampilan hasil. Menerima data hasil, status loading, dan error. Merender pesan loading/error atau data hasil. Bertanggung jawab untuk membangun data tree dan menampilkan visualisasi menggunakan react-d3-tree. Mengimplementasikan logika fitur live update visualisasi. Merender resep dalam format teks.
- Fungsi Layanan API (searchService.js):
 - **findRecipes(target, algo, mode, maxRecipes)**
Mengirim permintaan pencarian ke backend API /api/search.
 - **getElementImageURL(elementName)**
Membuat URL untuk mengambil gambar elemen melalui backend proxyc /api/image.
- Fungsi Bantu Tampilan Hasil (SearchResults.jsx):
 - **isBaseElement(name)**
Helper untuk mengecek apakah elemen adalah elemen dasar.
 - **buildInitialElementNode()**
Membuat struktur data node awal untuk elemen (sebelum resepnya ditambahkan).
 - **buildElementNodeRecursive()**
Fungsi rekursif untuk membangun struktur tree dari data resep untuk visualisasi statis.
 - **buildFullTreeData()**
Menginisialisasi proses pembangunan tree statis untuk satu jalur resep.
 - **renderNodeWithImage()**
Fungsi kustom untuk merender tampilan setiap node visualisasi tree, termasuk menampilkan gambar elemen.
 - **startLiveUpdate()**
Menginisialisasi state untuk memulai animasi live update visualisasi.

- **buildTreeForNextStep()**
Fungsi untuk membangun data tree untuk langkah live update berikutnya.
- **renderStep(), renderPath()**
Fungsi untuk merender resep dalam format tekstual.

- **Hooks React (useState, useEffect, useCallback)**

Digunakan di App.jsx dan SearchResults.jsx untuk mengelola state komponen, menjalankan efek samping (seperti memanggil API atau memulai timer animasi), dan mengoptimalkan kinerja dengan memoisasi fungsi.

C. Komunikasi Frontend-Backend

Frontend berkomunikasi dengan backend melalui HTTP requests:

- Frontend (melalui searchService.js) mengirimkan HTTP GET request ke endpoint /api/search di backend dengan parameter pencarian yang dikumpulkan dari SearchForm.
- Frontend (melalui SearchResults.jsx yang memanggil getElementImageURL di searchService.js) mengirimkan HTTP GET request ke endpoint /api/image di backend untuk mendapatkan data gambar elemen.
- Backend memproses permintaan, menjalankan algoritma pencarian, dan mengembalikan data hasil dalam format JSON ke frontend.
- Backend juga mengambil data gambar dari sumber eksternal dan mengirimkannya ke frontend melalui proxy /api/image.

4.2 Penjelasan Tata Cara Penggunaan Program

4.2.1 Antarmuka Awal

Saat pertama kali membuka aplikasi di browser, pengguna akan disambut dengan tampilan awal. Tampilan ini menampilkan logo aplikasi dan pesan "Click To Start". Ini adalah layar pembuka sebelum antarmuka utama muncul. Pengguna perlu mengklik di area yang ditentukan untuk melanjutkan ke antarmuka utama.

4.2.2 Antarmuka Utama

Setelah mengklik layar awal, antarmuka aplikasi akan bertransisi ke tampilan utama. Lalu akan ada kotak dialog dan dibawahnya akan muncul Form Pencarian. Tahap pencarian adalah sebagai berikut:

1. Form pencarian adalah bagian interaktif utama bagi pengguna untuk menentukan kriteria pencarian. Form ini terdiri dari elemen-elemen. Input Elemen Target, sebuah text input field di mana pengguna memasukkan nama elemen Little

Alchemy 2 yang resepnya ingin dicari. Contoh: "Mud", "Human", "Energy", dll. Input ini wajib diisi.

2. Pengguna bisa memilih Algoritma. Dan muncul Sekelompok buttons yang memungkinkan pengguna memilih algoritma pencarian yang akan digunakan.
 - BFS (Breadth-First Search). Algoritma ini akan mencari jalur resep dari elemen dasar menuju elemen target.
 - DFS(Depth-First Search) Algoritma ini akan mencari satu jalur resep dari elemen dasar menuju elemen target, namun tidak menjamin jalur yang ditemukan adalah yang terpendek.
 - Bidirectional (BDS). Algoritma ini akan mencari jalur dengan menjalankan dua pencarian secara bersamaan, satu dari elemen dasar ke depan dan satu dari elemen target ke belakang, bertemu di tengah. Bertujuan untuk menemukan jalur terpendek secara lebih efisien.
3. Setelah memilih algortima, akan muncul Pilihan Mode Pencarian (Mode) Sepasang buttons untuk menentukan jenis hasil yang diinginkan, Singlet: Program akan mencari hanya satu jalur resep. Lalu Multiple, Program akan mencari beberapa jalur resep yang berbeda untuk membuat elemen target (menggunakan algoritma yang dipilih).
4. Pada jumlah Resep Maksimal (Max Recipes), input berupa angka (number input field) yang hanya akan muncul dan dapat diisi ketika mode "Multiple" dipilih. Pengguna memasukkan angka positif yang menentukan berapa banyak jalur resep berbeda yang diinginkan program untuk dicari (misalnya, 5, 10, dst.). Jika mode "Multiple" dipilih tetapi input ini kosong atau nol, program akan menampilkan peringatan.
5. Tombol "Cari Resep" Akan memulai proses pencarian berdasarkan input yang telah dimasukkan. Setelah mengisi nama elemen target dan memilih opsi algoritma serta mode pencarian, pengguna mengklik tombol "Cari Resep". Aplikasi akan mulai memproses permintaan di backend.
6. Setelah backend selesai memproses permintaan, hasil akan ditampilkan di sisi kanan antarmuka (Panel Hasil). Selama proses pencarian berlangsung, panel ini akan menampilkan indikator loading
7. Jika terjadi kesalahan selama pencarian (misalnya, elemen target tidak valid, backend error, atau elemen target tidak dapat dibuat), panel hasil akan menampilkan pesan error yang relevan, seringkali disertai dengan gambar "Path Not Found".
8. Jika elemen target yang dimasukkan ternyata adalah salah satu dari 4 elemen dasar, panel hasil akan menampilkan pesan yang mengindikasikan bahwa elemen tersebut adalah elemen dasar dan tidak memerlukan resep.
9. Jika jalur ditemukan, informasi statistik mengenai proses pencarian akan ditampilkan, yaitu, Jumlah node (elemen atau kombinasi) yang dikunjungi atau

dievaluasi oleh algoritma selama pencarian dan Waktu yang dibutuhkan algoritma untuk menyelesaikan pencarian.

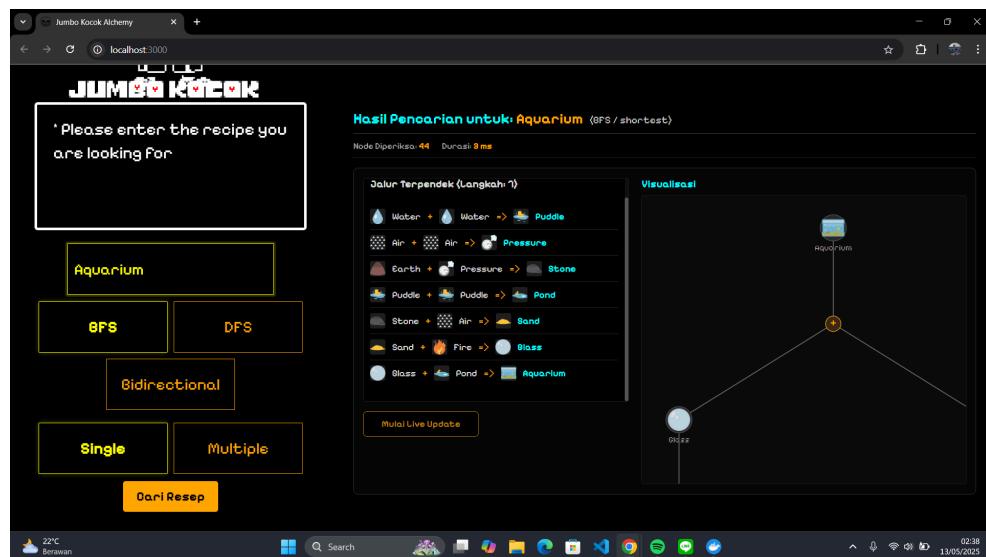
10. Program akan menampilkan visualisasi grafis dari resep dalam bentuk tree. Setiap node dalam tree merepresentasikan sebuah elemen atau kombinasi resep. Node elemen dasar akan berada di bagian paling bawah atau "daun" (leaf) tree membentuk jalur dari elemen dasar hingga elemen target di bagian paling atas.
11. Untuk setiap jalur resep yang ditampilkan, akan ada tombol khusus. Jika diklik, visualisasi tree untuk jalur tersebut akan mulai dibangun secara bertahap, langkah demi langkah, sesuai dengan urutan resep yang diperlukan dari elemen dasar. Akan ada sedikit delay antar setiap langkah animasi agar perubahannya terlihat jelas. Tombol akan menunjukkan status

4.3 Hasil Pengujian

4.3.1 Uji BFS

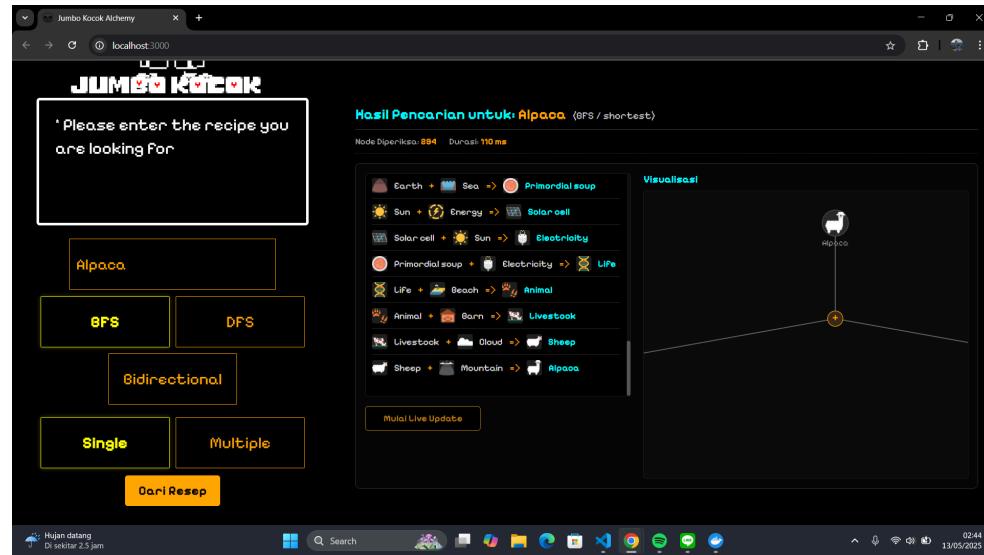
A. Single Path

- Tier 5



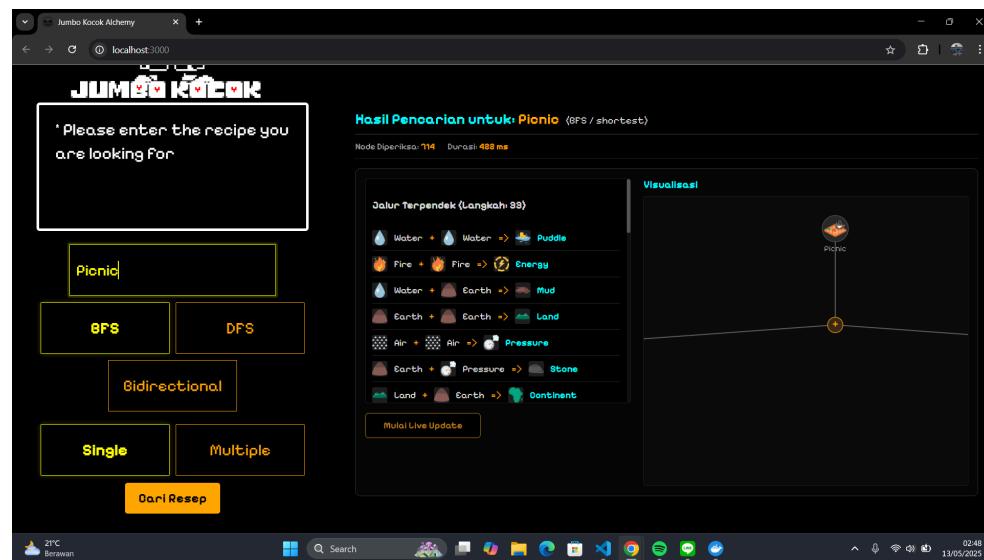
Gambar 4.3.1.1 BFS Single Path Tier 5

- Tier 10



Gambar 4.3.1.2 BFS Single Path Tier 10

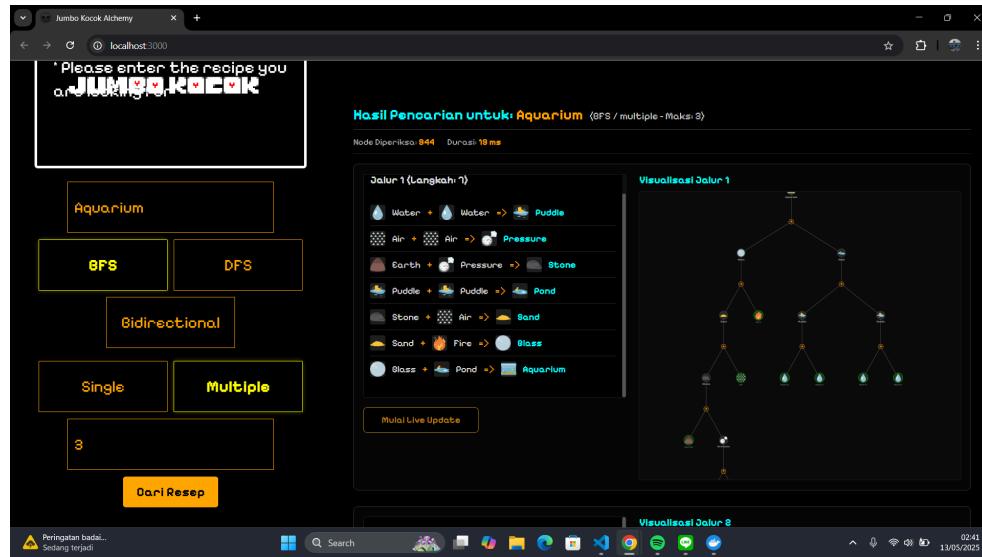
- Tier 15



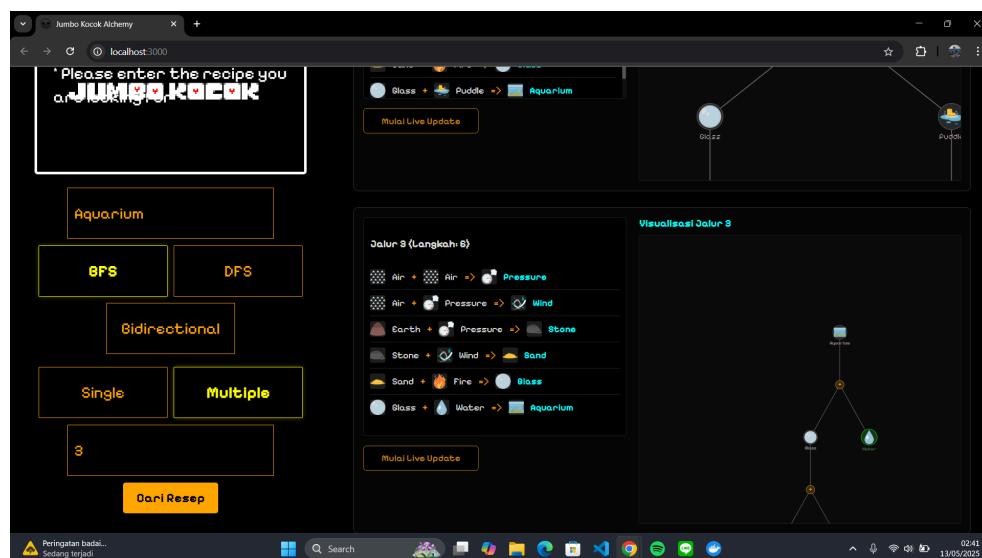
Gambar 4.3.1.3 BFS Single Path Tier 15

B. Multiple Paths

- Tier 5

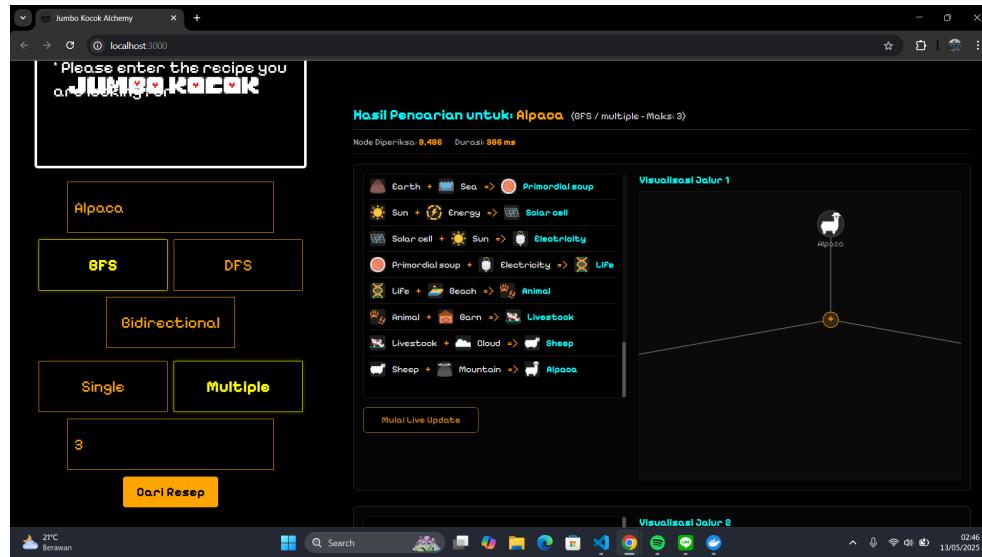


Gambar 4.3.1.4 BFS Multiple Paths Tier 5

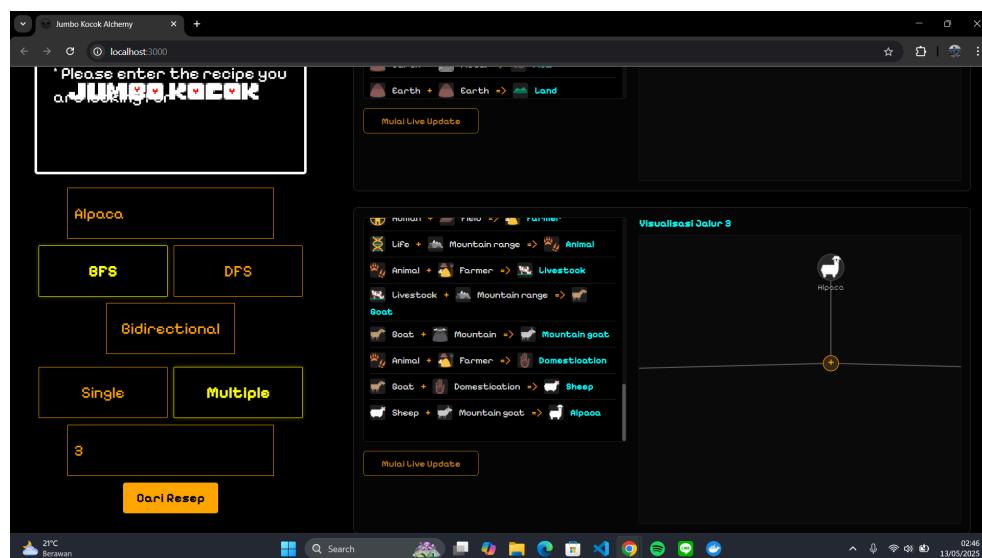


Gambar 4.3.1.5 BFS Multiple Paths Tier 5

- ## - Tier 10

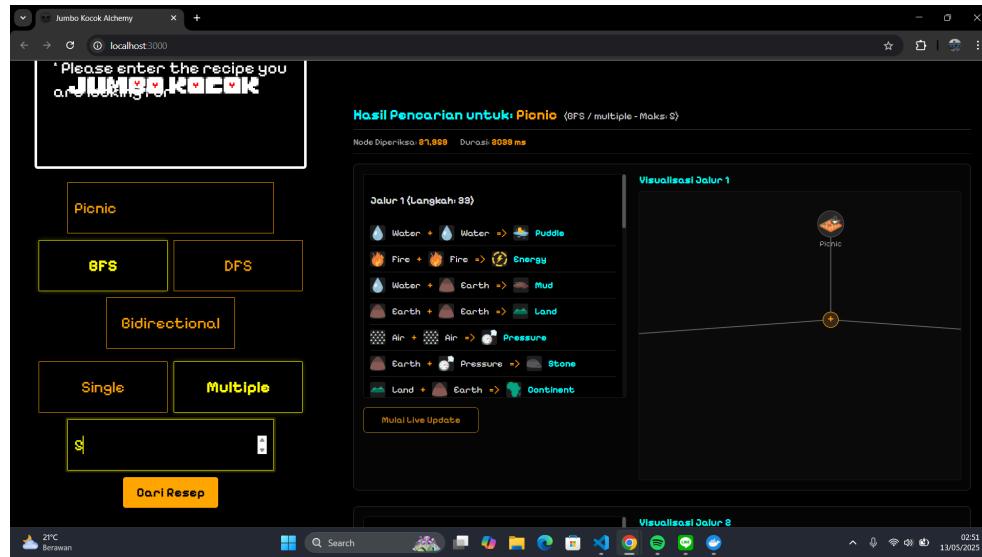


Gambar 4.3.1.6 BFS Multiple Paths Tier 10

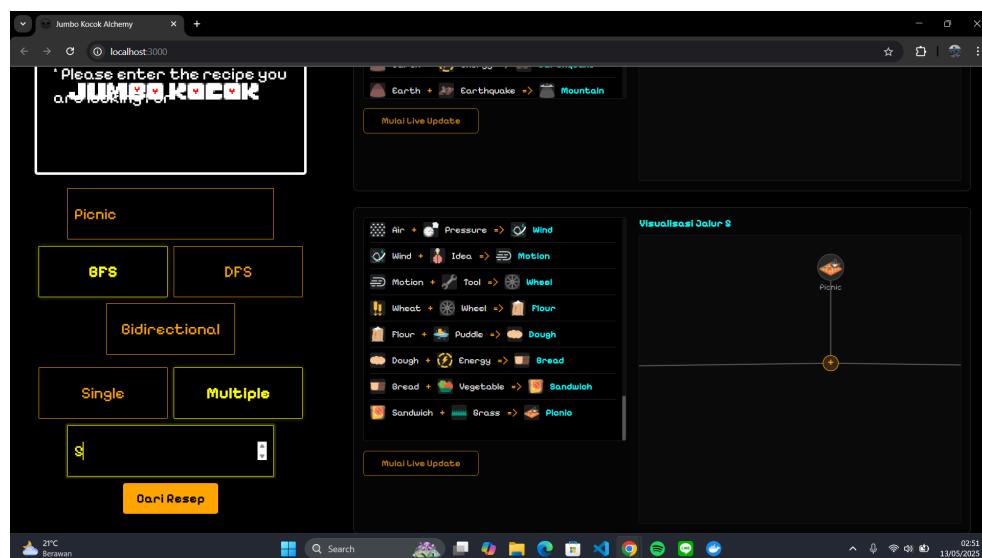


Gambar 4.3.1.7 BFS Multiple Paths Tier 10

- Tier 15



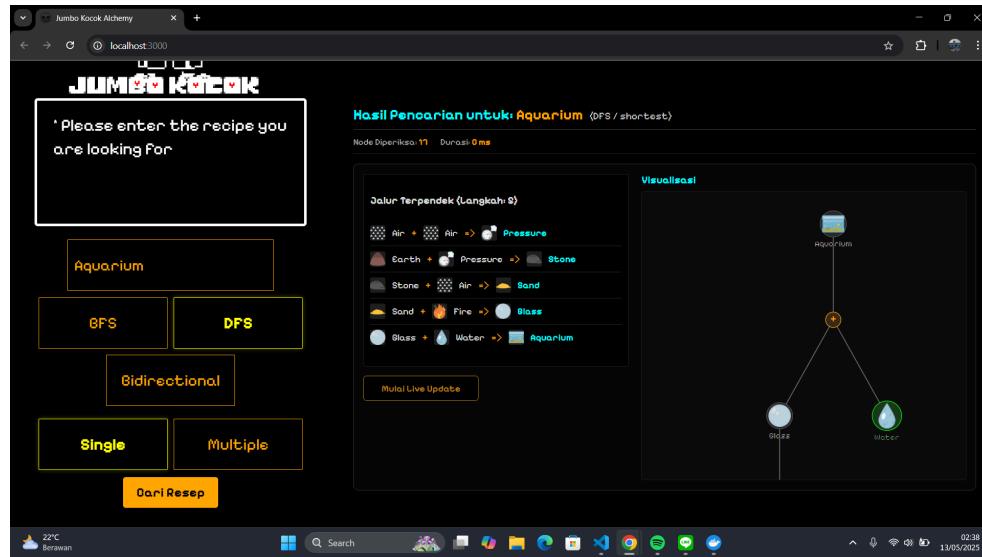
Gambar 4.3.1.8 BFS Multiple Paths Tier 15



Gambar 4.3.1.9 BFS Multiple Paths Tier 15

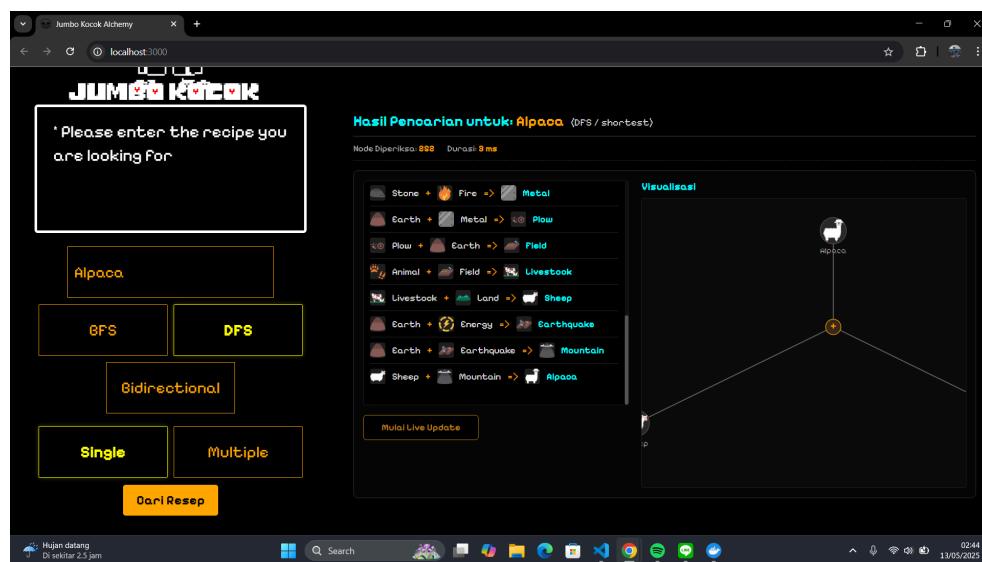
4.3.2 Uji DFS

- A. Single Path
 - Tier 5



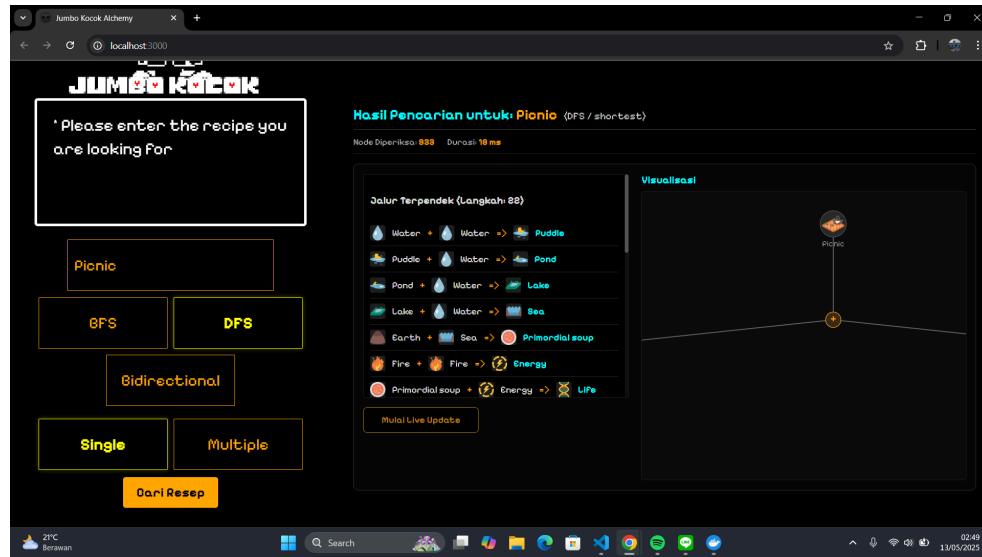
Gambar 4.3.2.1 DFS Single Path Tier 5

- Tier 10



Gambar 4.3.2.2 DFS Single Path Tier 10

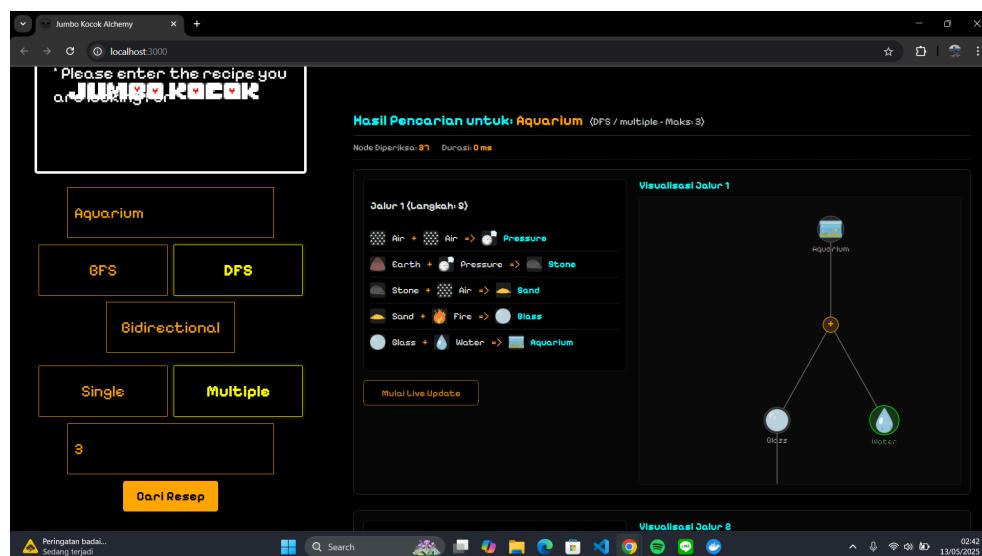
- Tier 15



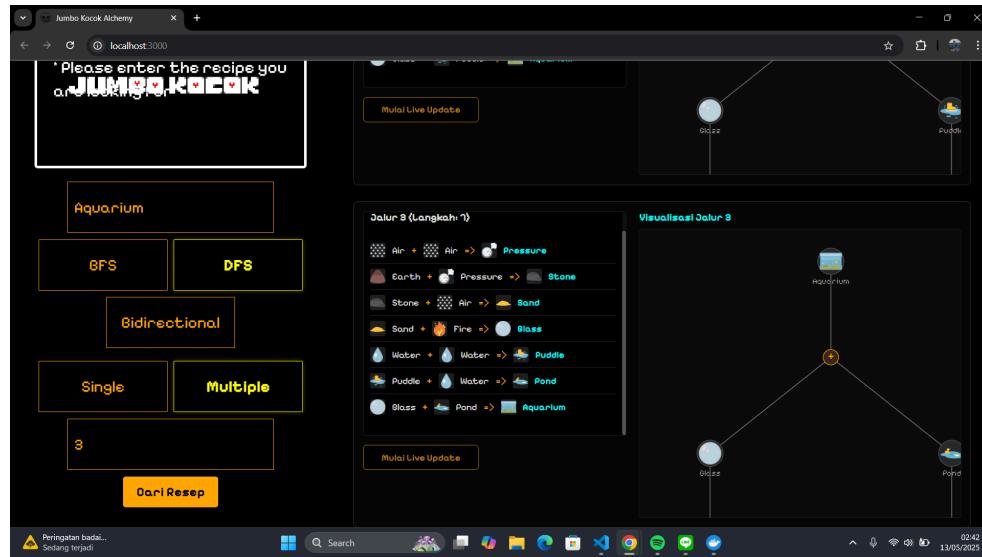
Gambar 4.3.2.3 DFS Single Path Tier 15

B. Multiple Paths

- Tier 5

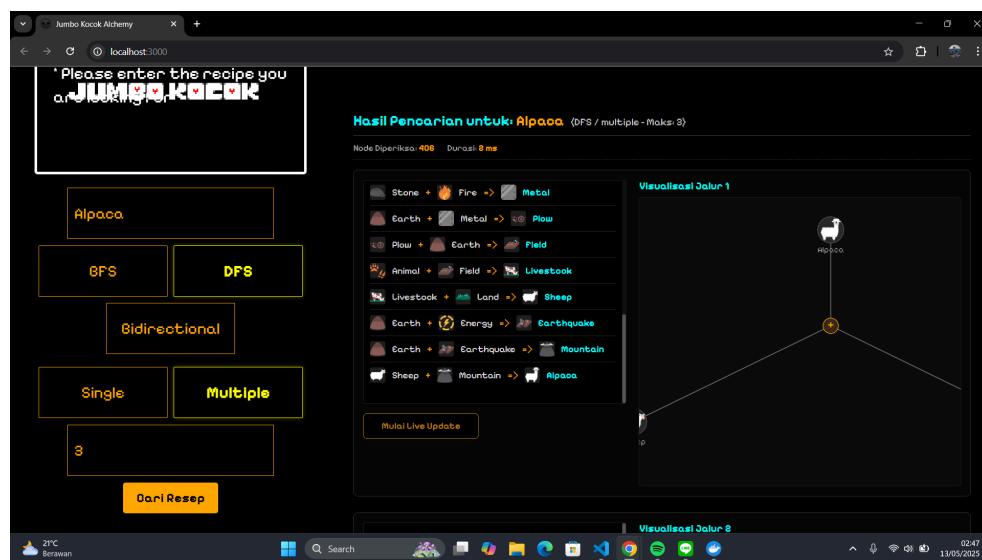


Gambar 4.3.2.4 DFS Multiple Paths Tier 5

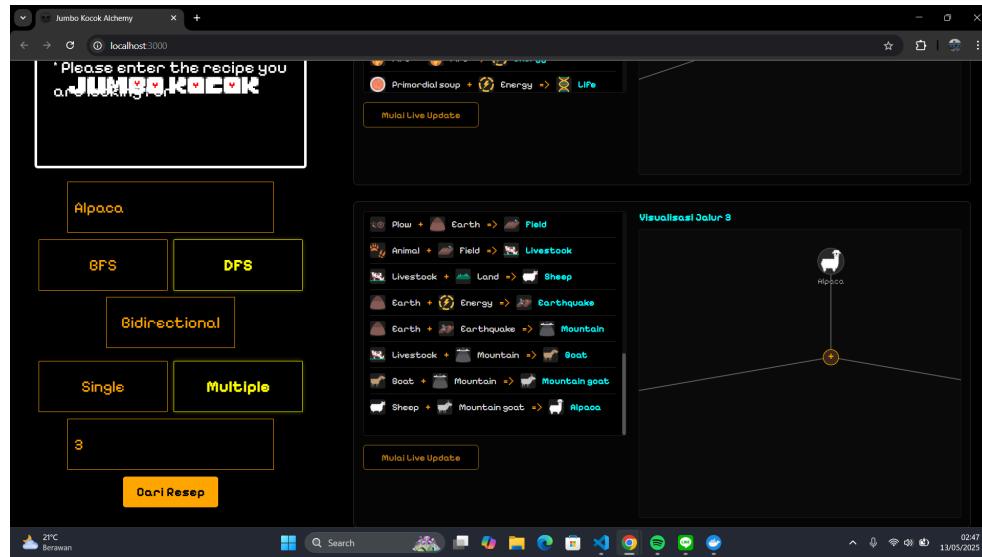


Gambar 4.3.2.5 DFS Multiple Paths Tier 5

- Tier 10

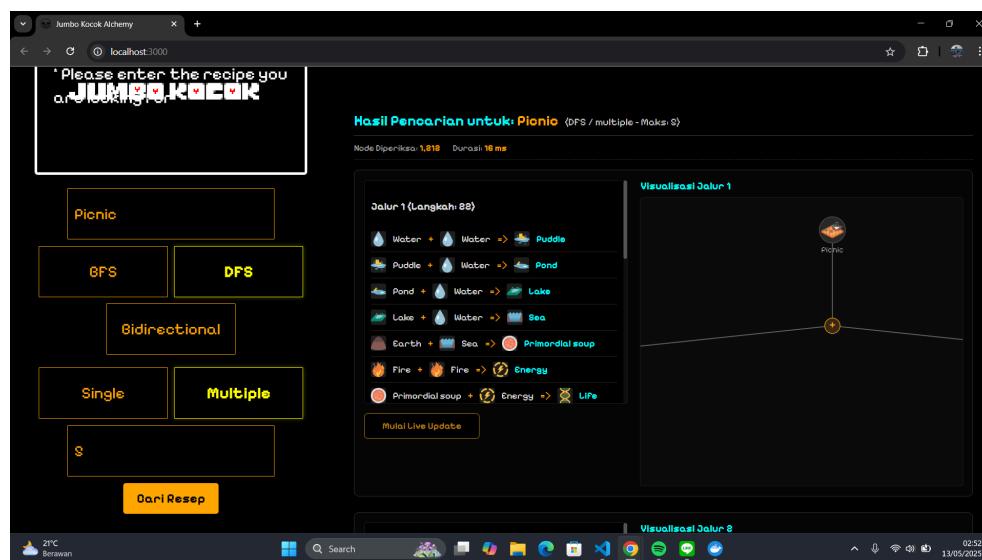


Gambar 4.3.2.6 DFS Multiple Paths Tier 10

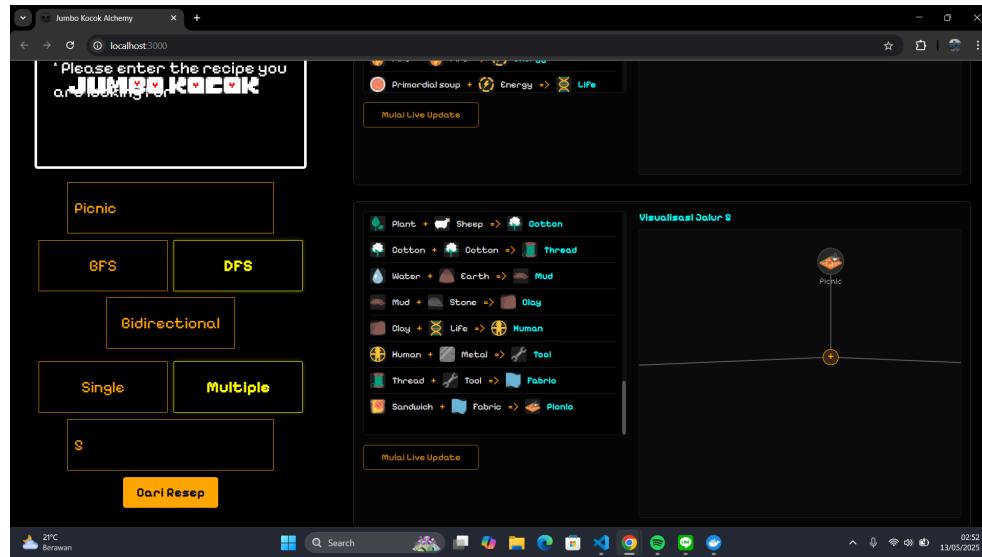


Gambar 4.3.2.7 DFS Multiple Paths Tier 10

- Tier 15



Gambar 4.3.2.8 DFS Multiple Paths Tier 15

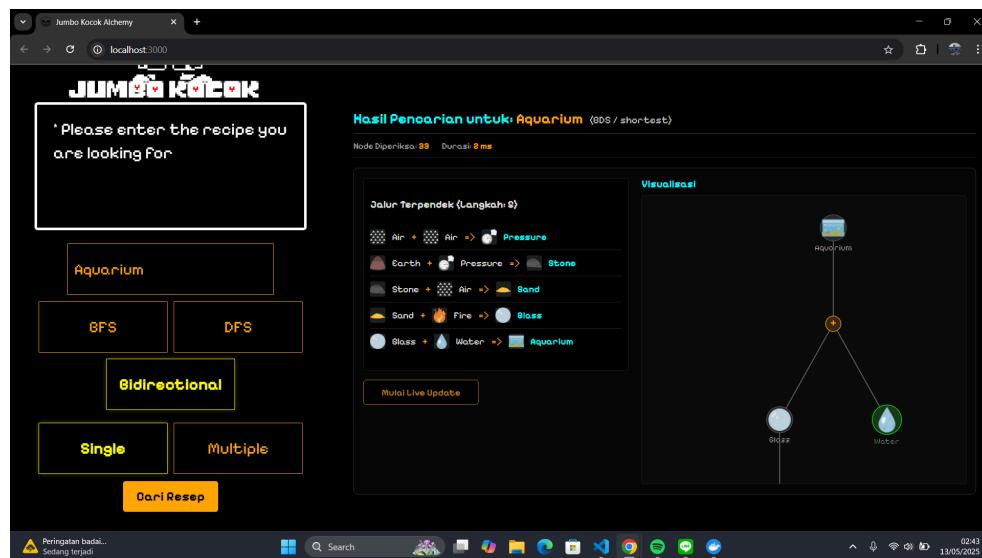


Gambar 4.3.2.9 DFS Multiple Paths Tier 15

4.3.3 Uji BDS

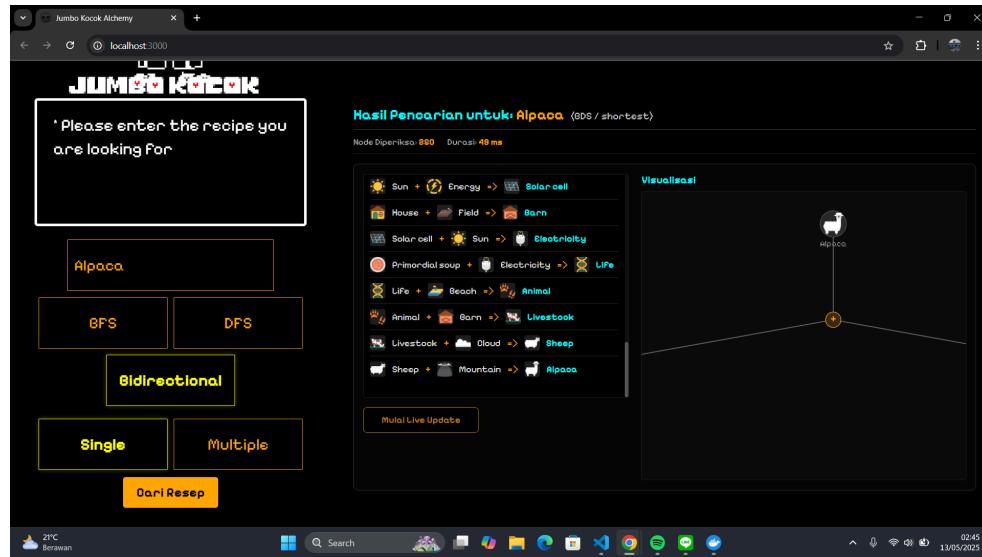
A. Single Path

- Tier 5



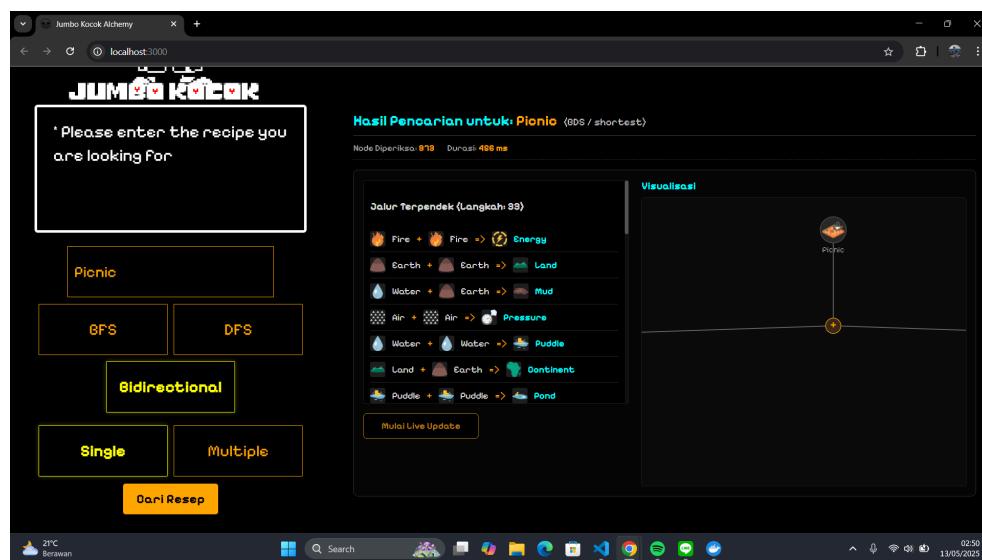
Gambar 4.3.3.1 BDS Single Path Tier 5

- Tier 10



Gambar 4.3.3.2 BDS Single Path Tier 10

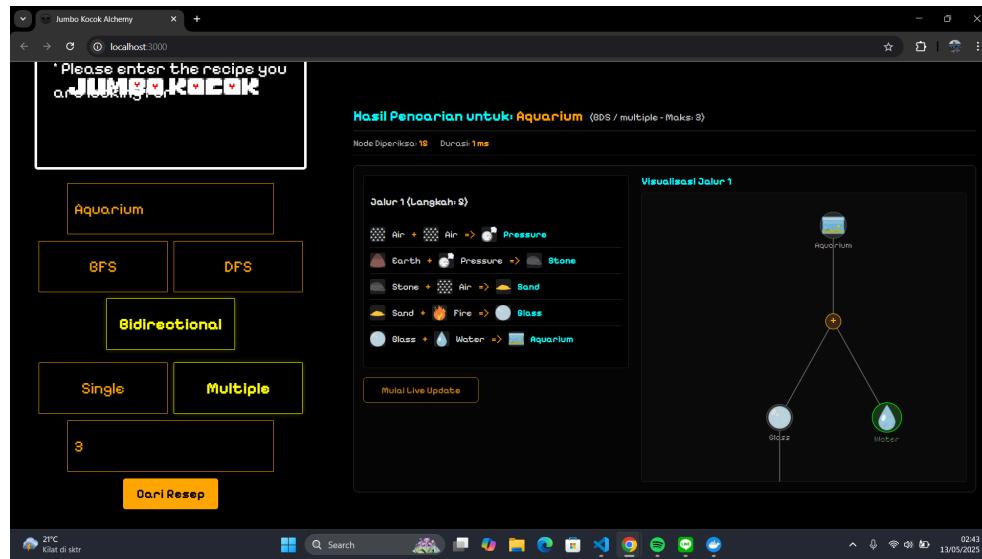
- Tier 15



Gambar 4.3.3.3 BDS Single Path Tier 15

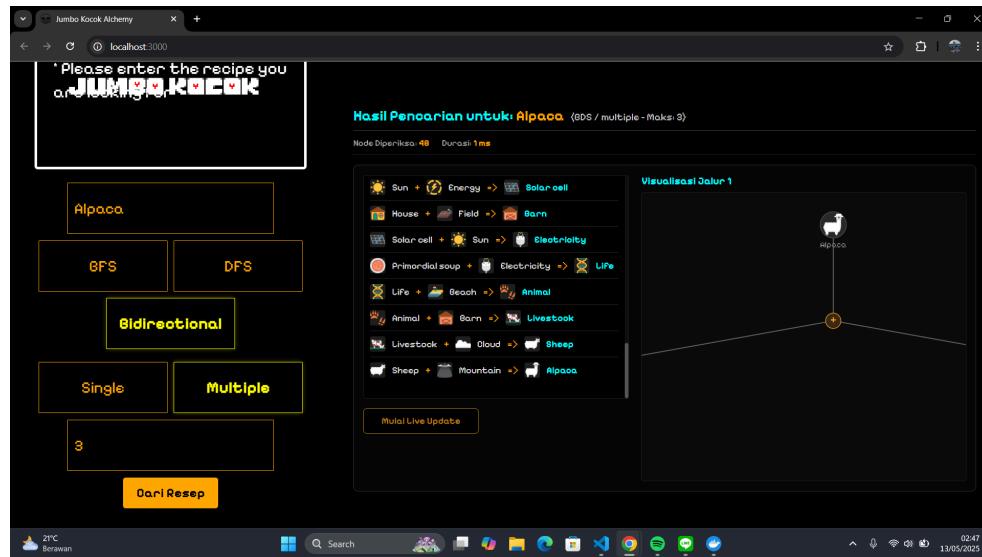
B. Multiple Paths

- Tier 5



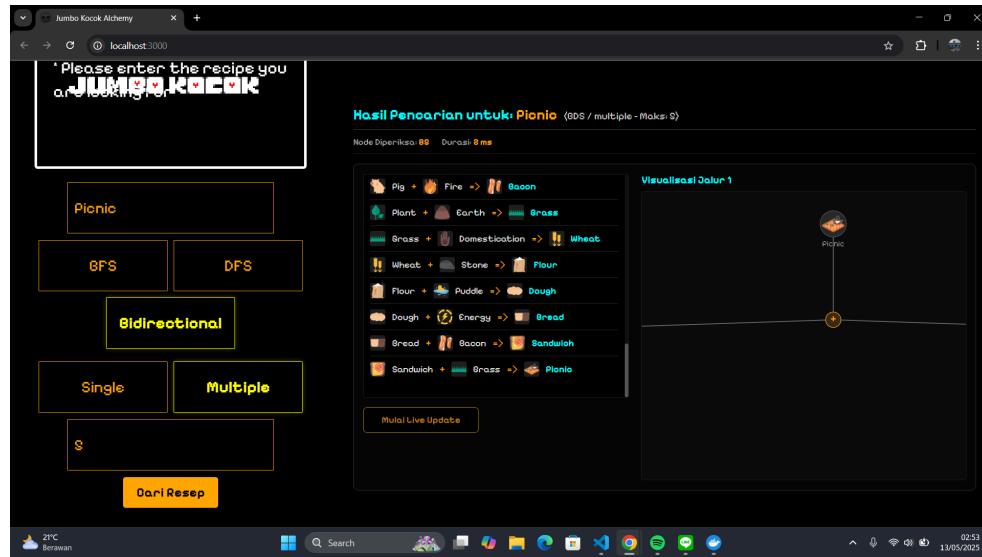
Gambar 4.3.3.4 BDS Multiple Paths Tier 5

- Tier 10



Gambar 4.3.3.5 BDS Multiple Paths Tier 10

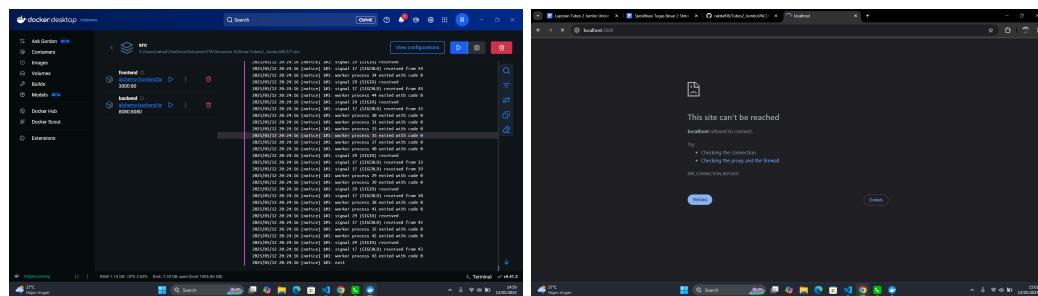
- Tier 15



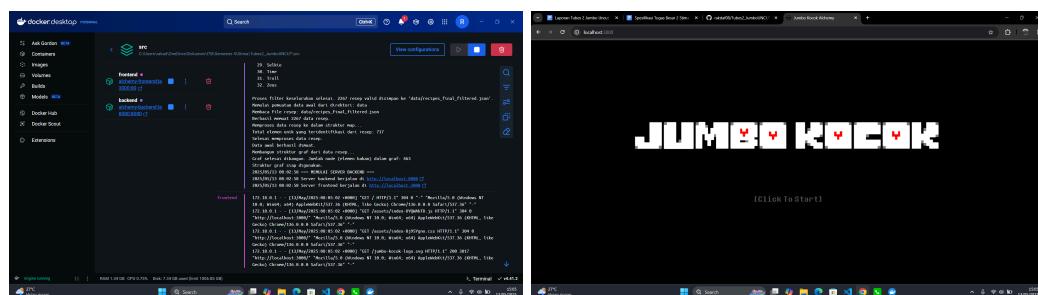
Gambar 4.3.3.6 BDS Multiple Paths Tier 15

4.3.4 Uji Bonus

A. Penggunaan Docker

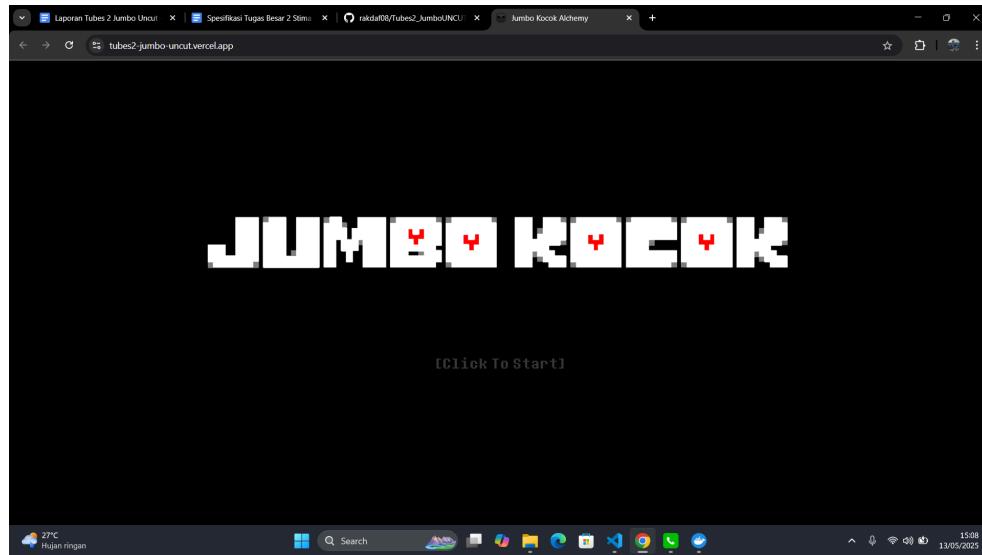


Gambar 4.3.4.1 Sebelum Docker dinyalakan (localhost:3000)



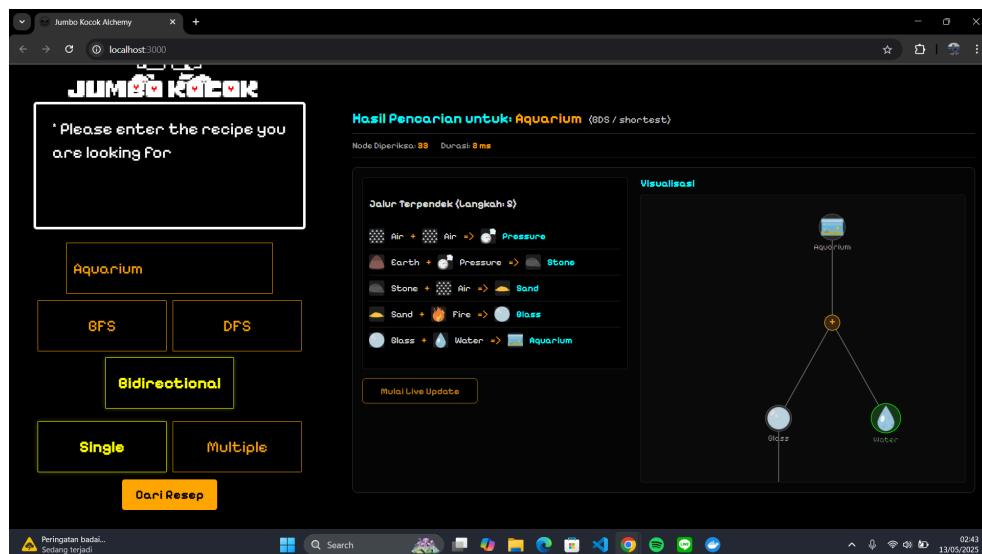
Gambar 4.3.4.2 Setelah Docker dinyalakan (localhost:3000)

B. Deploy Website



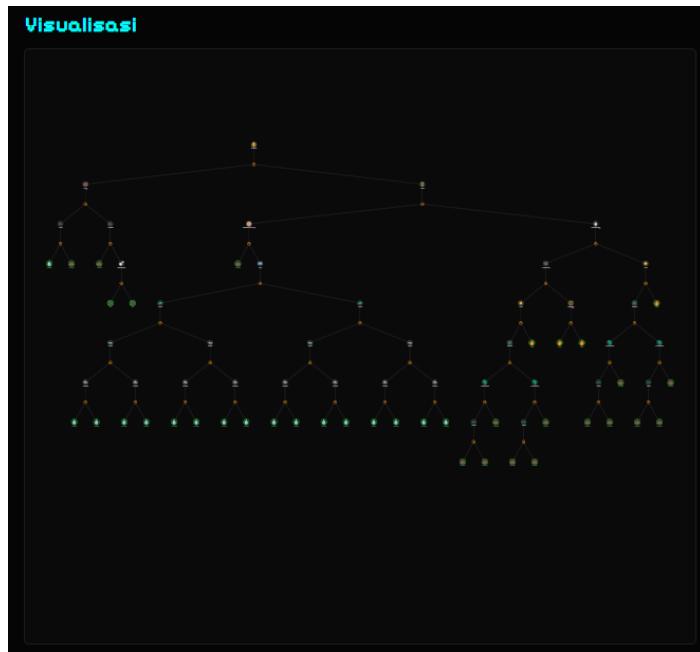
Gambar 4.3.4.3 Tampilan Awal Website (<https://tubes2-jumbo-uncut.vercel.app/>)

C. Penggunaan Bidirectional (BDS)



Gambar 4.3.4.4 Hasil Pencarian Menggunaan BDS

D. Penggunaan Live Update



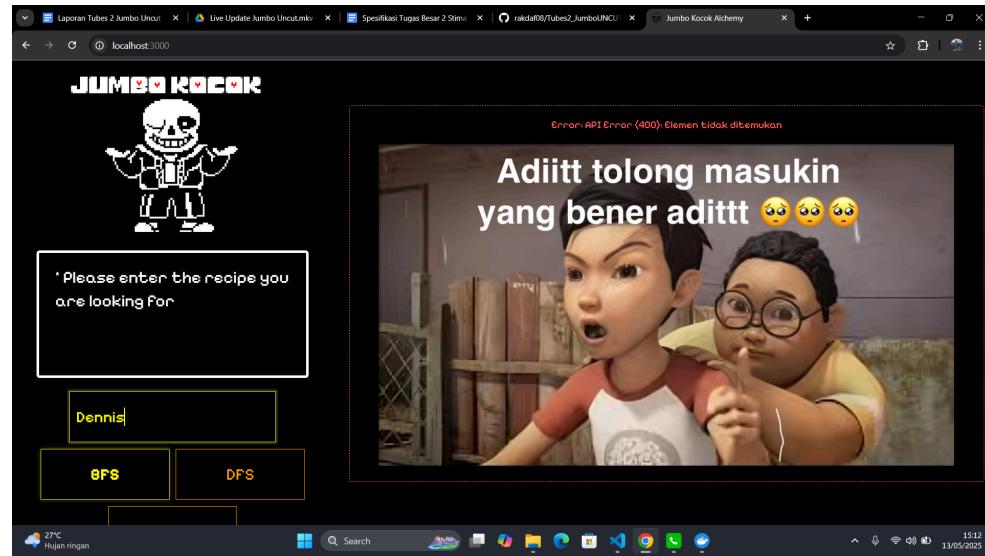
Gambar 4.3.4.4 Visualisasi Live Update

Video Live Update:

2025-05-13 14-25-31.mkv

4.3.5 Uji Error

A. Elemen Tidak Ada di JSON



Gambar 4.3.5.1 Hasil Pencarian Elemen yang Tidak Ada

4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang telah kami lakukan, terdapat empat hal yang bisa dihighlight. Pertama, pada BFS Single, semakin tinggi tier maka hampir semua node

diperiksa. Hal tersebut dikarenakan BFS akan mencari jalur menuju sebuah elemen target dan akan bekerja dengan level-order traversal dimana semua node pada level 1 diperiksa, kemudian level 2, dan seterusnya sampai menemukan elemen target. Hal tersebut berdampak untuk elemen dengan tier tinggi, BFS akan mengunjungi semua node terlebih dahulu yang membuat node diperiksa hampir semua.

Hal kedua adalah BDS Multiple hanya menampilkan satu path. Hal tersebut dikarenakan BDS pada dasarnya mencari satu titik temu tercepat antara pencarian maju (dari base) dan mundur (dari target). Karena struktur pencarian dan urutan traversalnya deterministik, hasilnya hampir selalu sama, kecuali ada randomisasi atau eksplorasi alternatif secara eksplisit.

Hal ketiga adalah pada MultiplePathsBFS node yang diperiksa jauh lebih banyak daripada BDS dan DFS. Hal tersebut dikarenakan untuk mencari beberapa path unik, kode melakukan banyak pencarian BFS dengan strategi berbeda dan kombinasi target resep yang berbeda, bahkan menggunakan banyak goroutine. Setiap pencarian BFS pada dasarnya melakukan eksplorasi dari awal, sehingga banyak node yang sama diperiksa berulang kali oleh worker yang berbeda. BFS juga cenderung mengeksplorasi semua kemungkinan kombinasi pada setiap level, sehingga jika ada banyak jalur alternatif, node visited count akan sangat tinggi. BDS dan DFS lebih fokus pada satu jalur atau lebih efisien dalam pruning (DFS dengan cache, BDS dengan dua arah), sehingga node yang diperiksa lebih sedikit.

Hal keempat adalah pada DFS (Single maupun Multiple), jumlah node yang diperiksa cenderung lebih sedikit dibandingkan BFS Multiple. Ini karena DFS langsung menyusuri jalur hingga dalam sebelum mundur, sehingga bisa menemukan target lebih cepat jika jalurnya tepat. Pendekatan ini hemat eksplorasi, apalagi jika elemen target berada di jalur awal yang ditelusuri. Namun, DFS tidak menjamin jalur terpendek. Pada Multiple Paths DFS, jalur yang ditemukan bisa lebih panjang karena DFS tidak mempertimbangkan panjang jalur saat pencarian. Hasil jalurnya juga sangat tergantung pada urutan eksplorasi elemen dasar. Dari sisi efisiensi, DFS lebih ringan memori dibanding BFS karena tidak menyimpan semua node di level yang sama. Dalam implementasi kami, penggunaan cache dan pembatasan kedalaman membantu DFS tetap efisien dan tidak berulang ke kombinasi yang sama.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

5.1 Kesimpulan

Melalui Tugas Besar 2 mata kuliah IF2211 Strategi Algoritma ini, kelompok kami berhasil membuat sebuah website yang mampu memecahkan permasalahan terkait kombinasi elemen yang diperlukan untuk mencapai elemen target pada permainan *Little Alchemy 2* dengan pendekatan algoritma graf traversal. Permasalahan akan dikonversi menjadi graf terarah, di mana simpul merepresentasikan elemen dan sisi merepresentasikan langkah dari bahan ke hasil berdasarkan resep yang ada.

Tiga algoritma pencarian yang kami gunakan adalah *Breadth First Search* (BFS), *Depth First Search* (DFS), dan *Bidirectional Search* (BDS) yang mana masing-masing memiliki keunggulan dan kelemahannya masing-masing. BFS unggul dalam mencari jalur terpendek, DFS bagus dalam mencari jalur alternatif, dan BDS dapat melakukan pencarian dengan pendekatan dua arah sehingga lebih efisien.

Penerapan konsep *multithreading* pada BFS dan DFS di mode *multiple path* juga terbukti dapat meningkatkan efisiensi secara signifikan, yang memungkinkan pencarian jalur dapat dilakukan secara paralel. Selain itu, pemanfaatan struktur data seperti map untuk cache, *parent tracking*, serta duplikasi *caching* di dalam program menjadi hal yang penting untuk menjaga performa dari aplikasi.

Di sisi lain, pengembangan frontend berbasis [React.js](#) juga memberikan nilai tambah terlebih dalam segi bentuk visualisasi yang interaktif, mulai dari tree statis hingga animasi *live update*. Hal ini bisa membuat pengguna lebih memahami jalur kombinasi elemen.

Dengan demikian, tugas besar ini tidak hanya menjadi latihan untuk mengimplementasikan algoritma yang sudah didapatkan di kelas saja, tetapi juga merupakan integrasi antara teori algoritma, pemrograman backend-frontend, visualisasi data, dan pengelolaan konkurensi.

5.2 Saran

Ada beberapa saran untuk pengembangan selanjutnya, di antaranya:

- Mengoptimalkan implementasi algoritma pencarian *Breadth First Search* (BFS) agar benar-benar dapat menampilkan jalur terpendek
- Menambahkan optimasi visualisasi tree agar bisa menangani elemen dengan jalur yang sangat panjang tanpa membebani frontend
- Memulai pengembangan program dengan lebih cepat

5.3 Refleksi

Awalnya, kami mengira bahwa tugas ini akan cukup mudah dikerjakan karena hanya terkait dengan algoritma pencarian graf seperti BFS dan DFS yang sudah dikerjakan di kelas. Akan tetapi pada kenyataannya implementasi nyata dalam bentuk website ini memunculkan beberapa tantangan yang membuat kami cukup kewalahan, mulai dari pengelolaan struktur data, pembuatan sistem multithreading yang aman dari *race condition*, integrasi visualisasi hasil pencarian dengan [React.js](#), hingga penggunaan docker dan pengintegrasinya agar dapat dibuat deployment sehingga bisa diakses di website secara online.

Meski sempat merasa tertekan, kami belajar banyak dari kesulitan-kesulitan yang kami hadapi. Kami jadi lebih memahami pentingnya arsitektur program yang baik, pemanfaatan cache untuk efisiensi, juga pengaturan state di frontend agar sinkron dengan backend. Pengalaman ini tentu mengasah kemampuan teknis kami, selain itu juga melatih kerja sama tim, manajemen waktu, dan disiplin dalam menyelesaikan proyek yang kompleks ini. Secara keseluruhan, tugas besar ini menjadi pengalaman belajar yang berharga untuk kami.

LAMPIRAN

Repository GitHub:

https://github.com/rakdaf08/Tubes2_JumboUNCUT

Link Deployment:

<https://tubes2-jumbo-uncut.vercel.app/>

Link Video:

<https://youtu.be/RYx1dd0j9wg>

Tabel Checklist Aplikasi Program

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi dapat memperoleh data <i>recipe</i> melalui scraping.	✓	
3	Algoritma <i>Depth First Search</i> dan <i>Breadth First Search</i> dapat menemukan <i>recipe</i> elemen dengan benar.	✓	
4	Aplikasi dapat menampilkan visualisasi <i>recipe</i> elemen yang dicari sesuai dengan spesifikasi.	✓	
5	Aplikasi mengimplementasikan multithreading.	✓	
6	Membuat laporan sesuai dengan spesifikasi.	✓	
7	Membuat bonus video dan diunggah pada Youtube.	✓	
8	Membuat bonus algoritma pencarian <i>Bidirectional</i> .	✓	
9	Membuat bonus <i>Live Update</i> .	✓	
10	Aplikasi di- <i>containerize</i> dengan Docker.	✓	
11	Aplikasi di- <i>deploy</i> dan dapat diakses melalui internet.	✓	

DAFTAR PUSTAKA

- Munir, R., & Maulidevi, N. U. (2025). *Breadth/Depth First Search (BFS/DFS) - Bagian 1.* Program Studi Teknik Informatika, STEI ITB.
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf)
- Munir, R., & Maulidevi, N. U. (2025). *Breadth/Depth First Search (BFS/DFS) - Bagian 2.* Program Studi Teknik Informatika, STEI ITB.
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf)
- Recloak. (2017). *Little Alchemy 2*. <https://littlealchemy2.com/>
- Fandom. (2024). *Little Alchemy 2 Wiki*. <https://little-alchemy-2.fandom.com/>
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.
- Tarjan, R. E. (1972). Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1(2), 146–160.
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.
- Lea, D. (2000). Concurrent programming in Java: Design principles and patterns (2nd ed.). Addison-Wesley.