Laporan Tugas Besar 2
IF2211 Strategi Algoritma

Pemanfaatan Algoritma BFS dan DFS dalam Pencarian Recipe pada Permainan Little Alchemy 2

Disusun oleh:
Kelompok 28 – *Minekrep*

Indah Novita Tangdililing	13523047
Bevinda Vivian	13523120
Naomi Risaka Sitorus	13523122

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

Kata Pengantar

Kami mengucapkan syukur kepada Tuhan Yang Maha Esa atas rahmat dan petunjuk-Nya telah mengarahkan saya untuk menyelesaikan laporan ini tepat waktu. Dokumen ini merupakan sebuah laporan yang disusun untuk memenuhi tugas besar 2 dalam mata kuliah IF2211 Strategi Algoritma. Selain itu, laporan ini juga bertujuan untuk meningkatkan pemahaman kami mengenai penerapan Algoritma *Breadth First Search* dan *Depth First Search*.

Terima kasih kami ucapkan kepada Bapak Dr. Ir. Rinaldi, M.T. yang telah menjadi dosen pengampu mata kuliah di kelas K-02 IF2211 Strategi Algoritma. Selain itu kami juga ingin mengungkapkan terima kasih kepada kakak-kakak asisten dari Laboratorium Ilmu dan Rekayasa Komputasi yang senantiasa membantu kami dalam Tugas Besar 2 Strategi Algoritma. Kami sadar bahwa laporan ini masih jauh dari kata sempurna. Oleh karena itu, jika terdapat kesalahan dalam penulisan atau ketidaksesuaian dalam materi yang kami sampaikan dalam laporan ini, kami memohon maaf. Tentunya kami juga sangat mengharapkan saran dan kritik yang membangun untuk meningkatkan kualitas tugas besar ataupun laporan ini.

Bandung, 13 Mei 2025

Kelompok Minekrep

Daftar Isi

Daftar Gambar

Bab 1

Deskripsi Tugas

Dalam Tugas Besar 2 Strategi Algoritma, mahasiswa diminta untuk menerapkan Pemanfaatan Algoritma BFS dan DFS dalam Pencarian Recipe pada Permainan Little Alchemy 2. Berikut ini deskripsi masalah lengkap yang telah diberikan kepada mahasiswa dalam Tugas Besar 2 Strategi Algoritma:

Gambar 1. Little Alchemy 2

(sumber: https://www.thegamer.com)

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 2

Landasan Teori

2.1 Dasar Teori

2.1.1 Penjelajahan Graf

Penjelajahan graf merupakan proses mengunjungi simpul-simpul dalam suatu graf dengan cara yang sistematis. Dalam ilmu komputer, graf sering digunakan sebagai representasi dari berbagai persoalan kompleks, seperti jaringan sosial, jaringan web, peta rute, dan berbagai sistem yang memiliki hubungan antar elemen. Terdapat dua pendekatan utama dalam merepresentasikan graf untuk proses pencarian solusi: graf statis yang telah terbentuk sebelum proses pencarian dimulai, dan graf dinamis yang terbentuk saat proses pencarian dilakukan.

Penjelajahan graf memiliki banyak aplikasi praktis, di antaranya pencarian rute, penelusuran situs web, eksplorasi *social network*, dan pencarian dokumen dalam struktur direktori. Teknik penjelajahan graf juga menjadi fondasi dari berbagai algoritma pencarian solusi berbasis graf, seperti pencarian tanpa informasi (*uninformed/blind search*) dan pencarian dengan informasi (*informed search*). Dalam konteks permainan seperti Little Alchemy 2, penjelajahan graf dapat dimanfaatkan untuk menemukan kombinasi elemen yang diperlukan untuk membentuk elemen target tertentu.

2.1.2 Algoritma *Breadth First Search*

Breadth First Search (BFS) atau pencarian melebar adalah algoritma penjelajahan graf yang mengunjungi simpul-simpul dalam graf dengan cara sistematis level per level. Algoritma ini dimulai dari simpul awal, kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu, sebelum bergerak ke level berikutnya. BFS menggunakan struktur data *queue* yang menerapkan prinsip FIFO (*First In First Out*) untuk menyimpan simpul yang telah dikunjungi.

Pada implementasinya, BFS menjamin ditemukannya solusi dengan jumlah langkah minimum atau optimal jika setiap langkah memiliki biaya yang sama. BFS memiliki kompleksitas waktu O(b), di mana b adalah faktor percabangan maksimum dan d adalah kedalaman solusi terbaik. Sedangkan kompleksitas ruangnya juga O(b^d), yang berarti algoritma ini membutuhkan memori yang cukup besar untuk menyimpan semua simpul pada level yang sama. Dalam konteks pencarian recipe di Little Alchemy 2, BFS akan mengeksplorasi semua kombinasi elemen yang dapat dibentuk dari elemen dasar secara berurutan berdasarkan jumlah langkah yang diperlukan, sehingga dapat menemukan cara pembuatan elemen target dengan jumlah kombinasi paling sedikit.

2.1.3 Algoritma *Depth First Search*

Depth First Search (DFS) atau pencarian mendalam adalah algoritma penjelajahan graf yang mengeksplorasi sejauh mungkin pada satu jalur sebelum mundur (backtrack) dan mencoba jalur lain. Algoritma ini dimulai dari simpul awal, lalu mengunjungi satu simpul tetangga dan melanjutkan pencarian secara rekursif ke simpul tersebut. Ketika mencapai simpul yang tidak memiliki tetangga yang belum dikunjungi, algoritma *backtracking* ke simpul sebelumnya dan mencoba jalur alternatif lainnya. DFS menggunakan struktur data tumpukan (stack) yang menerapkan prinsip LIFO (*Last In First Out*) atau memanfaatkan rekursi yang secara implisit menggunakan stack.

DFS menjamin ditemukannya solusi selama nilai faktor percabangan terbatas dan ada penanganan untuk jalur berulang. Namun, DFS tidak menjamin solusi optimal atau jalur terpendek. Kompleksitas waktu DFS adalah O(b^m), di mana b adalah faktor percabangan dan m adalah kedalaman maksimum dari ruang status. Sedangkan kompleksitas ruangnya adalah O(bm), yang lebih efisien dibandingkan BFS dalam hal penggunaan memori. Karakteristik *backtracking* dalam DFS sangat berguna untuk menyelesaikan persoalan yang memiliki banyak alternatif pilihan. Dalam konteks Little Alchemy 2, DFS akan mengeksplorasi kombinasi elemen secara mendalam hingga mencapai elemen target atau sampai tidak ada kombinasi valid lagi, kemudian mencoba jalur alternatif lainnya.

2.1.4 Algoritma *Bidirectional Search*

Bidirectional Search adalah algoritma pencarian graf yang bekerja dengan memulai dua pencarian sekaligus: satu dari simpul awal (*source*) ke arah simpul tujuan (*goal*), dan satu lagi dari simpul tujuan ke arah simpul awal. Pendekatan ini secara signifikan dapat mengurangi ruang pencarian karena kedalaman eksplorasi dari masing-masing sisi hanya setengah dari pencarian satu arah.

Dalam konteks permainan *Little Alchemy 2*, pencarian resep suatu elemen dapat dimodelkan sebagai pencarian lintasan dalam graf, di mana simpul-simpul merepresentasikan elemen dan sisi-sisinya mewakili kombinasi dua elemen yang menghasilkan elemen baru. Menggunakan *Bidirectional Search* untuk menemukan resep, berarti tidak hanya menelusuri kombinasi dari elemen dasar ke target (forward search), tetapi juga mencoba memecah elemen target menjadi komponen penyusunnya dari arah berlawanan (backward search). Pendekatan ini dapat mempercepat pencarian resep secara signifikan, terutama untuk elemen kompleks, dengan menghentikan eksplorasi saat kedua arah pencarian bertemu di suatu elemen perantara.

2.2 Aplikasi Web

Aplikasi web untuk pencarian recipe pada permainan Little Alchemy 2 dibangun dengan arsitektur client-server yang terdiri atas frontend dan backend. Frontend dikembangkan menggunakan JavaScript dengan framework Next.js atau React.js yang berfokus pada antarmuka pengguna dan visualisasi hasil pencarian recipe. Backend dikembangkan menggunakan bahasa pemrograman Golang yang bertanggung jawab untuk melakukan web scraping data elemen dari situs Fandom Little Alchemy 2, serta implementasi algoritma BFS dan DFS untuk mencari recipe.

Aplikasi web ini memungkinkan pengguna untuk memilih algoritma pencarian yang diinginkan (BFS atau DFS), menentukan apakah mencari recipe terpendek atau multiple recipe, dan memasukkan parameter banyak recipe maksimal yang ingin dicari. Hasil pencarian divisualisasikan dalam bentuk tree yang menunjukkan

kombinasi elemen yang diperlukan dari elemen dasar hingga elemen target. Untuk meningkatkan performa, aplikasi memanfaatkan multithreading dalam mode pencarian multiple recipe. Komunikasi antara frontend dan backend dilakukan melalui API REST, di mana frontend mengirimkan permintaan pencarian recipe dan backend mengembalikan hasil pencarian untuk divisualisasikan.

Bab 3

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Masalah yang perlu diselesaikan dalam Tugas Besar 2 ini adalah mencari resep atau rute untuk membentuk elemen yang diinginkan dari keempat elemen dasar, yaitu *fire*, *water*, *air*, dan *earth*. Pengguna dapat mencari satu atau lebih resep menggunakan algoritma pencarian *Breadth First Search (BFS)* atau *Depth First Search (DFS)*.

Pencarian resep diawali dengan masukan dari pengguna berupa elemen target yang ingin dicari resepnya. Jika elemen target merupakan salah satu dari elemen dasar, maka pencarian tidak diperlukan dan elemen tersebut akan langsung dikembalikan sebagai hasil. Setiap elemen yang tidak termasuk dalam elemen dasar akan digabungkan dengan elemen lainnya berdasarkan resep yang ada.

Algoritma akan membangun graf yang menggambarkan hubungan antara elemen-elemen tersebut. Setiap simpul pada graf akan merepresentasikan elemen, dan setiap sisi akan menghubungkan elemen-elemen yang dapat digabungkan berdasarkan resep. Dalam algoritma BFS, pencarian elemen secara iteratif, sedangkan algoritma DFS akan mencari jalur secara rekursif.

Setelah pencarian selesai, apabila elemen target ditemukan, ditampilkan hasil berupa resep yang membentuknya sebanyak permintaan pengguna. Jika tidak ditemukan, akan ditampilkan pesan bahwa elemen tersebut tidak ditemukan di resep manapun. Selain itu, ditampilkan waktu pencarian dan banyak simpul yang dikunjungi.

3.2 Pemetaan Masalah Menjadi Elemen-Elemen Algoritma DFS dan BFS

Dalam program pencarian ini, kami memecahkan masalah pencarian resep dengan memetakan data ke dalam struktur graf yang dapat ditelusuri dengan algoritma DFS dan BFS. Elemen-elemen dalam game dimodelkan sebagai node dalam graf, sementara resep pembentukan menjadi edge yang menghubungkan antar node. Struktur tier diimplementasikan untuk mengoptimalkan pencarian, dengan elemen dasar berada pada tier 1 dan elemen kompleks pada tier yang lebih tinggi berdasarkan kompleksitas pembuatannya.

Pada algoritma DFS, penelusuran berfokus pada eksplorasi mendalam satu jalur resep hingga menemukan elemen dasar atau menemukan siklus, lalu melakukan backtracking. Implementasi ini cocok untuk mencari semua variasi resep dan digabungkan dengan paralelisme menggunakan goroutine untuk performa lebih baik.

Sementara pada BFS, kami menggunakan queue untuk melacak elemen yang akan dieksplorasi secara level demi level, memprioritaskan eksplorasi resep dengan tier rendah terlebih dahulu. Kedua algoritma menerapkan deteksi siklus berdasarkan tier, di mana resep yang menghasilkan elemen dengan tier lebih rendah atau sama akan dilewati, mencegah rekursi tak terbatas dan stack overflow. Struktur RecipeTree digunakan untuk merepresentasikan hasil pencarian, membentuk pohon hirarkis dari elemen target hingga elemen-elemen dasar yang dibutuhkan untuk membuatnya.

3.3 Fitur Fungsional

Website ini mampu menemukan langkah-langkah kombinasi elemen dari Little Alchemy 2 untuk menghasilkan elemen target yang dimasukkan pengguna. Pengguna dapat memilih algoritma pencarian yang dipakai, yaitu BFS atau DFS, serta jumlah resep yang diinginkan. Hasil pencarian ditampilkan secara hirarkis dalam bentuk pohon resep. Waktu pencarian, jumlah node (elemen) yang dikunjungi, dan jumlah solusi yang ditemukan akan ditampilkan. Resep-resep diperoleh dari *scraping website* Wiki Little Alchemy 2 yang berlangsung secara statis, yaitu saat

pertama kali *website* dijalankan dan disimpan dalam suatu file berformat JSON. Pada website ini juga kami menerapkan fitur bonus yang sudah disebutkan pada spesifikasi tugas besar sebelumnya, selanjutnya akan kami dijelaskan terkait fitur bonus yang sudah kami kerjakan.

3.3.1 Docker

Minekrep dapat dijalankan dengan menggunakan Docker, baik untuk backend maupun frontend. Dengan memanfaatkan Docker dan docker-compose, nantinya pengguna yang mengakses dari CLI hanya perlu menjalankan satu perintah yaitu docker-compose up --build pada direktori src untuk membangun serta menjalankan seluruh layanan aplikasi secara lokal. Proses pada docker ini memastikan konsistensi dalam pengembangan, mempermudah instalasi, serta mempercepat proses deployment dan pengujian.

3.3.2 Deployment

Untuk memberikan akses yang lebih luas, Minekrep berhasil dideploy secara daring dan bisa diakses melalui internet. Backend aplikasi dihosting menggunakan platform Railway. API yang tersedia dapat diakses secara publik melalui https://deployminekrep-production.up.railway.app dengan beberapa endpoint utama, dengan metode GET untuk api seperti /api/elements untuk mengambil semua elemen, /api/elements/basic untuk elemen dasar, dan /api/search untuk melakukan pencarian resep. Khusus untuk /api/search hanya diperbolehkan metode POST, yaitu hasil dari pencarian yang ada pada frontend. Terkait frontend, Minekrep juga telah deploy frontend dengan menggunakan layanan Vercel, yang mendukung build pipeline otomatis serta optimalisasi kinerja frontend untuk produksi. Aplikasi web ini dapat diakses secara publik di https://minekrep-2025.vercel.app/, sehingga pengguna dapat langsung mengakses *interface* aplikasi melalui link rtersebut tanpa perlu instalasi lokal.

3.3.3 Algoritma Bidirectional

Minekrep juga menyediakan strategi pencarian Depth-First Search (DFS) dan Breadth-First Search (BFS), namun juga mengimplementasikan algoritma Bidirectional Search. Fitur ini menjadi opsi tambahan pada *interface* yang bisa

dipilih secara langsung oleh pengguna. Algoritma ini bekerja dengan pendekatan dua arah, yakni pencarian dari target ke bawah (*forward*) dan dari elemen dasar ke atas (*backward*) secara paralel. Apabila kedua jalur pencarian bertemu pada titik elemen yang sama, maka jalur tersebut digunakan untuk menyusun *recipe tree* yang lebih efisien. Pendekatan ini mempercepat pencarian dan sangat membantu dalam kasus target yang kompleks.

3.3.4 Live Update Visualization

Fitur Live Update melakukan visualisasi interaktif terhadap proses pencarian elemen secara real-time. Fitur ini ditujukan agar pengguna tidak hanya melihat hasil akhir dari pencarian, tetapi juga dapat memahami bagaimana proses pencarian berlangsung. Visualisasi ini disusun dalam bentuk tree menggunakan React Flow di sisi frontend, dan diperbarui secara berkala berdasarkan progres dari pencarian di sisi backend. Fungsi TrackLiveUpdate() diterapkan pada setiap algoritma di backend yang digunakan pada setiap langkah pencarian, termasuk dalam DFS, BFS, maupun Bidirectional, dan berfungsi sebagai jembatan antara logika pencarian dan visualisasi *interface* pengguna.

3.4 Arsitektur Aplikasi Web

3.4.1 Frontend

Frontend dari aplikasi web dibangun dengan framework Next.js serta React dalam bahasa pemrograman CSS, JavaScript, dan JSX (JavaScript XML). *Landing page* menyediakan pilihan bagi pengguna untuk melakukan pencarian atau melihat profil kelompok. Fitur yang tersedia di halaman pencarian ialah memilih algoritma, memasukkan elemen target, dan memasukkan banyak resep yang ingin dicari. Kolom hasil pencarian menampilkan pohon resep dan statistik pencarian. Pada halaman profil, pengguna dapat melihat informasi lebih detail mengenai kelompok Minekrep dan anggotanya.

3.4.2 Backend

Backend dari aplikasi web dibangun menggunakan bahasa pemograman Go (Golang). *Backend* bertanggung jawab untuk melakukan *scraping* data resep dan menyimpannya ke dalam file JSON, mencari resep menggunakan

algoritma BFS atau DFS, dan menyediakan *endpoint* REST API untuk mengirim hasil pencarian ke *frontend* untuk visualisasi.

Backend memiliki struktur sebagai berikut:

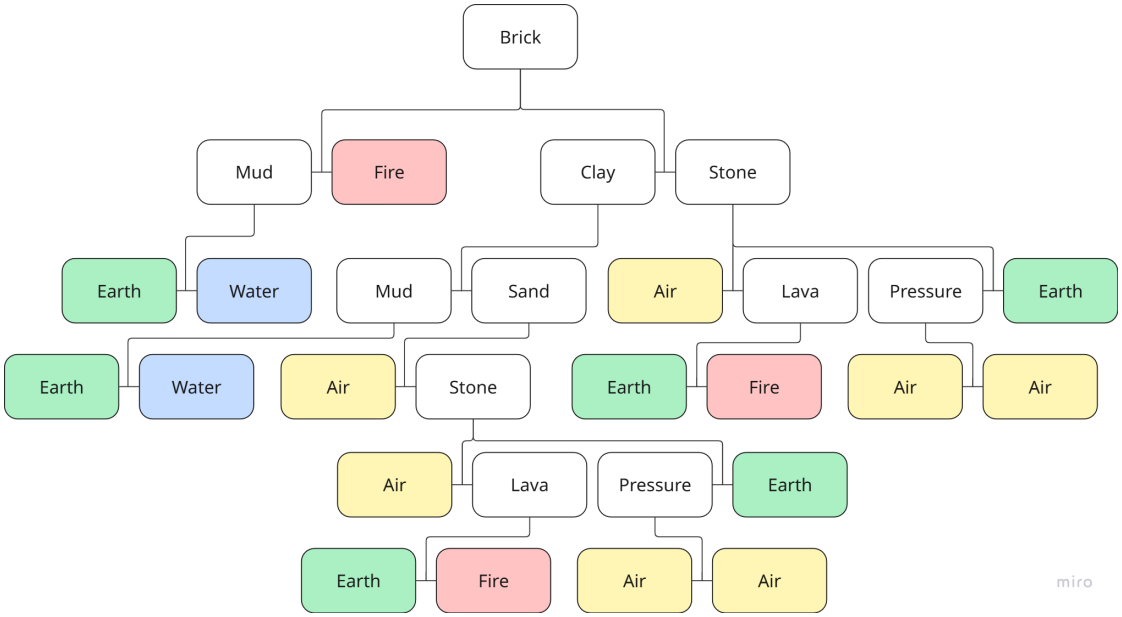
```
backend
│   Dockerfile          # Menciptakan container backend untuk Docker
│   go.mod
│   go.sum
│   main.go             # File utama pemanggil scraper dan fungsi pencarian
├───api                 # Mengirim informasi dari backend ke frontend
│       └─handlers.go
├───data                # Menyimpan hasil scraping resep
│       └─recipes.json      (terbentuk setelah dilakukan scraping)
├───scraper             # Mengekstrak data resep dari website ke JSON
│       └─scrap.go
├───searchalgo          # Algoritma pencarian resep
│       ├─ bfs.go
│      ├─ bidirectional.go
│      └─ dfs.go
└───utilities           # Struktur data dan helper function  untuk pencarian
        ├─ models.go
        └─ utils.go
```

3.5 Contoh Ilustrasi Kasus

Dalam pencarian resep untuk elemen "Brick" menggunakan algoritma DFS, sistem pertama mencari di database dan menemukan bahwa Brick dapat dibuat melalui kombinasi Mud + Fire atau Clay + Stone. DFS memilih salah satu jalur terlebih dahulu, misalnya Mud + Fire. Dari database, DFS kemudian kembali dan memeriksa Mud yang dapat dibuat dari Water + Earth, lalu mengunjungi node Fire yang sudah merupakan Base Element. Algoritma kemudian melakukan backtracking karena komponen Clay ber-tier lebih tinggi dari target. Kedua komponen tersebut adalah elemen dasar sehingga eksplorasi cabang Mud selesai. Selama proses ini, TrackLiveUpdate mencatat dan mengirimkan setiap langkah eksplorasi ke frontend, memungkinkan visualisasi real-time bagaimana algoritma menelusuri berbagai jalur dan Algoritma akhirnya menemukan solusi lengkap: Brick

dapat dibuat dari kombinasi elemen dasar dengan urutan langkah resep yang valid, yaitu Mud + Fire dengan Mud yang terbuat dari Water + Earth.

Untuk elemen "Brick" menggunakan algoritma BFS, pendekatan yang diambil berbeda. BFS mulai dengan mencari semua resep yang menghasilkan Brick: Mud + Fire dan Clay + Stone. BFS tidak langsung mengeksplorasi Mud secara mendalam, melainkan menyimpan semua komponen level saat ini (Fire, Clay Stone, Earth, Water) dalam queue. BFS kemudian mengambil komponen pertama dalam queue dan menelusuri semua cara pembuatannya. Elemen-elemen baru ditambahkan ke queue. BFS menandai Mud sebagai teratasi dan menyimpan jalur pembuatannya. Berikutnya adalah Fire yang sudah merupakan elemen dasar. Dengan cara ini, BFS menjelajahi semua elemen secara level demi level, dengan syarat elemen harus dengan tier lebih rendah dari target. Pendekatan ini memastikan BFS menemukan solusi dengan jumlah langkah paling sedikit terlebih dahulu.



DFS Path: Brick -> Mud -> Earth -> Water -> Fire, lalu backtrack saat ingin mengunjungi Clay

Tabel BFS

Iterasi	V	Queue	Brick	Mud	Fire	Earth	Water

0	Brick	Mud, Fire, Clay, Stone	T	F	F	F	F
1	Mud	Fire, Clay, Stone, Earth, Water	T	T	F	F	F
2	Fire	Clay, Stone, Earth, Water	T	T	T	F	F
3	Clay	[BACKTRACK] Earth, Water	T	T	T	F	F
4	Earth	Water	T	T	T	T	F
5	Water	-	T	T	T	T	T

Bab 4

Implementasi dan Pengujian

4.1 Teknis Program *Scraping*

4.1.1 Struktur Data

Struktur Data	Penjelasan

type Recipe struct {
Element1 string `json:"element1"`
Element2 string `json:"element2"`
Result string `json:"result"`
IconFilename string
`json:"icon_filename"`
}
	Menyimpan informasi tentang resep, yakni dua elemen yang digabungkan (Element1 dan Element2), hasilnya (Result), serta nama file ikon elemen hasil

4.1.2 Fungsi dan Prosedur

Fungsi/Prosedur	Penjelasan
ScrapeIfNeeded(filepath)	Melakukan *scraping* semua elemen Little Alchemy 2, kecuali *myths and monsters*, jika file recipes.json belum ada
getElementLinks(url, baseURL)	Mengambil semua link elemen valid dari halaman utama wiki
scrapeElementPage(url, mythsSet)	Mengekstrak resep per halaman elemen
saveToJSON(data, filename)	Menyimpan data yang diekstrak ke recipes.json

4.2 Teknis Program Pencarian Secara Umum

4.2.1 Struktur Data

Struktur Data	Penjelasan

type Recipe struct {
Element1 string `json:"element1"`
Element2 string `json:"element2"`
Result string `json:"result"`
IconFilename string
`json:"icon_filename"`
}
	Menyimpan informasi tentang resep, yakni dua elemen yang digabungkan Element1 dan Element2), hasilnya (Result), serta nama file ikon elemen hasil

type Element struct {
Name string
Tier int
}
	Menyimpan informasi tentang elemen, yaitu nama elemen dan tingkatannya (Tier)

type Node struct {
Element string
Path []string
Visited map[string]bool
Depth int
Ingredients map[string][]string
}
	Merepresentasikan sebuah simpul dalam pencarian resep, yang mencakup elemen yang sedang diproses, jalur pencarian (Path), status kunjungan elemen lainnya (Visited), kedalaman pencarian (Depth), dan daftar bahan yang digunakan untuk membuat elemen tersebut (Ingredients)

type RecipeTree struct {
Element string `json:"element"`
Ingredients []RecipeTree
`json:"ingredients,omitempty"`
}
	merepresentasikan pohon resep, di mana setiap elemen memiliki bahan-bahan (elemen sebelumnya) yang dapat digunakan untuk membuat elemen tersebut

type Step struct { Current string `json:"current"` Queue []string `json:"queue"` Element1 string `json:"element1"` Element2 string `json:"element2"` Result string `json:"result"`	Menyimpan langkah-langkah dalam pencarian resep, termasuk elemen yang sedang diproses (Current), antrian elemen berikutnya (Queue), serta informasi tentang

}	elemen yang digunakan dalam langkah tersebut

type ResultStep struct {
Element1 string `json:"element1"`
Element2 string `json:"element2"`
Result string `json:"result"`
IconFilename string `json:"icon_filename"`
}
	Mewakili langkah hasil dalam pencarian resep, dengan informasi tentang elemen yang digunakan dan elemen yang dihasilkan

type RecipeResult struct {
TargetElement string `json:"targetElement"`
Steps []ResultStep `json:"steps"`
}
	Menyimpan hasil pencarian resep, termasuk elemen target yang ingin dicapai dan langkah-langkah (ResultStep) yang diperlukan untuk mencapainya

type LiveUpdateStep struct {
Step int `json:"step"`
Message string `json:"message"`
PartialTree *RecipeResult `json:"partial_tree,omitempty"`
HighlightNodes []string `json:"highlight_nodes"`
}
	Menyimpan informasi tentang langkah-langkah yang sedang berjalan dalam pencarian resep *real-time*, dengan pembaruan status dan informasi tentang elemen yang sedang diproses, digunakan untuk bonus *live update*

4.2.2 Fungsi dan Prosedur

Fungsi/Prosedur	Penjelasan
IsBaseElement(element string) bool	Mengembalikan *true* jika elemen merupakan elemen dasar (fire, water, air, dan earth)
BuildRecipeTree(element string, ingredients map[string][]string) RecipeTre	Membuat pohon resep untuk elemen dengan memetakan elemen dan bahan-bahan yang dibutuhkan

CalculateTreeDepth(tree RecipeTree) int	Mengembalikan kedalaman pohon resep
Max(a, b int) int	Mengembalikan nilai maksimum antara dua angka
IsSameRecipeTree(tree1, tree2 RecipeTree) bool	Mengembalikan *true* jika kedua pohon resep sama, termasuk urutan bahan-bahannya
CopyMap(original map[string][]string) map[string][]string	Menyalin peta resep
initializeTiers[]	Menginisialisasi tingkat (tier) untuk setiap elemen berdasarkan resep yang tersedia
LoadRecipes(filePath string)	Memuat resep dari file JSON dan menyimpannya dalam peta resep
SetLiveUpdateCallback(callback func(element string, path []string, found map[string][]string))	Membuat fungsi *callback* untuk memperbarui progres pencarian resep
TrackLiveUpdate(element string, path []string, found map[string][]string)	Memanggil fungsi *callback* untuk melakukan *live update* dalam proses pencarian
FindIconForRecipe(element1, element2, result string) string	Mencari ikon gambar untuk resep berdasarkan elemen yang digunakan dan hasilnya

4.3 Teknis Program Pencarian Berbasis BFS

4.3.1 Struktur Data

Menggunakan struktur data pencarian secara umum.

4.3.2 Fungsi dan Prosedur

Fungsi/Prosedur	Penjelasan
BFSSearch(target, maxRecipes)	Mencari pohon resep dari elemen target menggunakan algoritma BFS, yaitu pencarian secara iteratif per level, yang diproses secara *multithreading* jika jumlah resep yang dicari (maxRecipes) lebih dari satu
processRecipe(e1 string, e2 string, found map[string][]string, visitCount *int, steps *[]utilities.Step) bool	Memproses BFS satu kombinasi elemen untuk membentuk target dan mengembalikan *true* jika semua bahan bisa dibuat atau merupakan elemen dasar

4.4 Teknis Program Pencarian Berbasis DFS

4.4.1 Struktur Data

Menggunakan struktur data pencarian secara umum serta tambahan struktur data sebagai berikut.

Struktur Data	Penjelasan

type SafeCounter struct {
  v int
  mux sync.Mutex
}
	Menyimpan jumlah elemen yang dikunjungi secara *thread-safe* di seluruh *goroutine* DFS untuk *multithreading*

4.4.2 Fungsi dan Prosedur

Fungsi/Prosedur	Penjelasan
DFSSearch(target, maxRecipes)	Mencari pohon resep dari elemen target menggunakan algoritma DFS, yaitu pencarian secara rekursif hingga

	mencapai target atau simpul daun yang tidak bisa diekspansi lagi, yang diproses secara *multithreading* jika jumlah resep yang dicari (maxRecipes) lebih dari satu
ExploreAllCombinations(e1, e2 string, baseMap map[string][]string, results *[]map[string][]string, counter *SafeCounter)	Mengeksplorasi semua kombinasi bahan dari e1 dan e2 menggunakan DFS
ExploreElementRecipes(element string, currentMap map[string][]string, counter *SafeCounter) []map[string][]string	Melakukan DFS untuk mendapatkan semua peta resep yang valid untuk satu elemen
(*SafeCounter).Inc()	Melakukan *increment* counter secara *thread-safe*
(*SafeCounter).Value()	Mengambil nilai dari counter secara *thread-safe*

4.5 Teknis Program Utama

4.5.1 Struktur Data

Tidak ada struktur data yang didefinisikan di program utama.

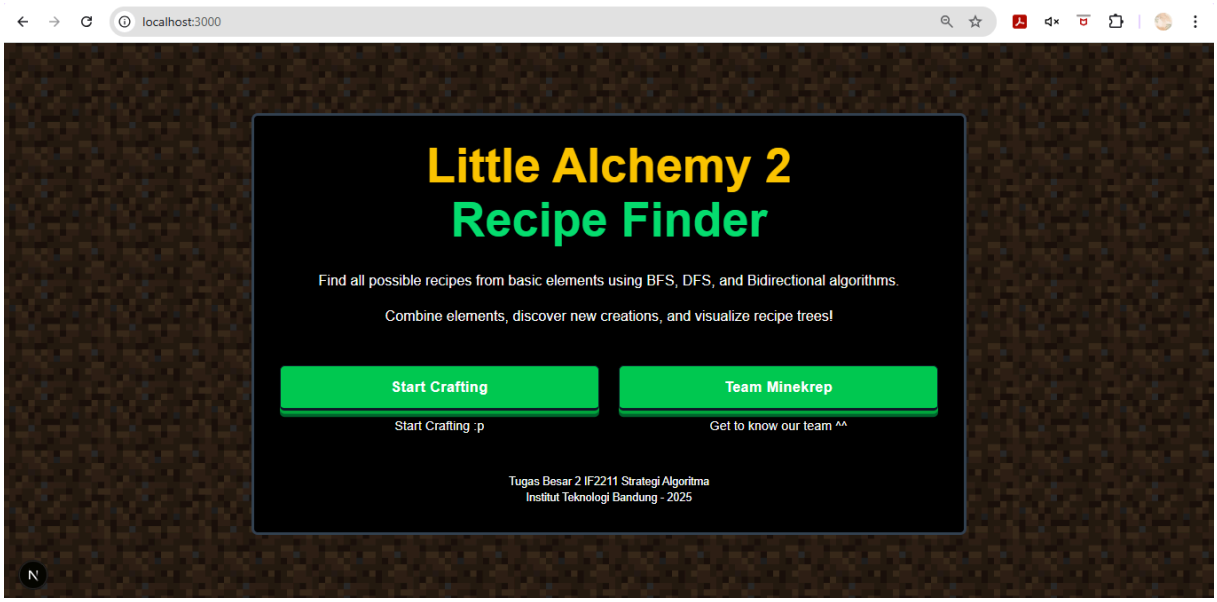
4.5.2 Fungsi dan Prosedur

Fungsi/Prosedur	Penjelasan
main()	Fungsi utama program yang memanggil prosedur *scraping*, *load* data dari recipes.json, dan menjalankan server
runServer(port)	Mengatur *handler* HTTP untuk API dan *frontend*, lalu menjalankan HTTP server di port tertentu

4.6 Tata Cara Penggunaan Program

4.6.1 *Landing Page*

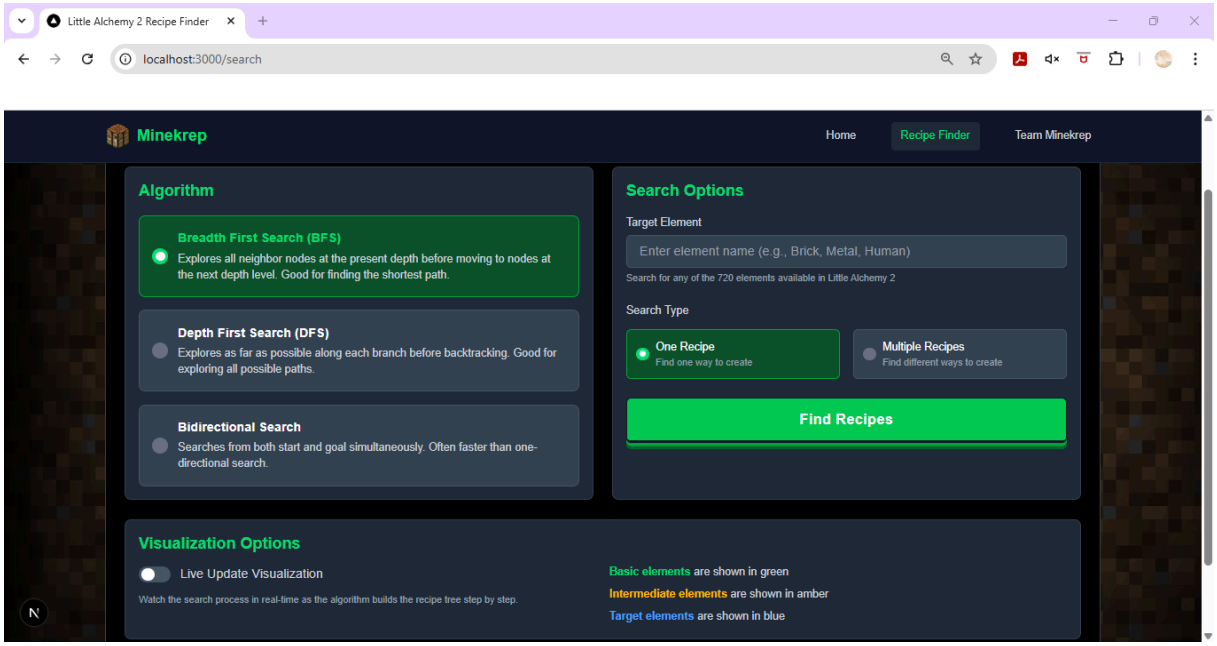
Landing page berisi informasi utama tentang *website* Little Alchemy 2 Recipe Finder. Pengguna dapat memilih untuk melakukan pencarian resep atau melihat profil kelompok Minekrep.



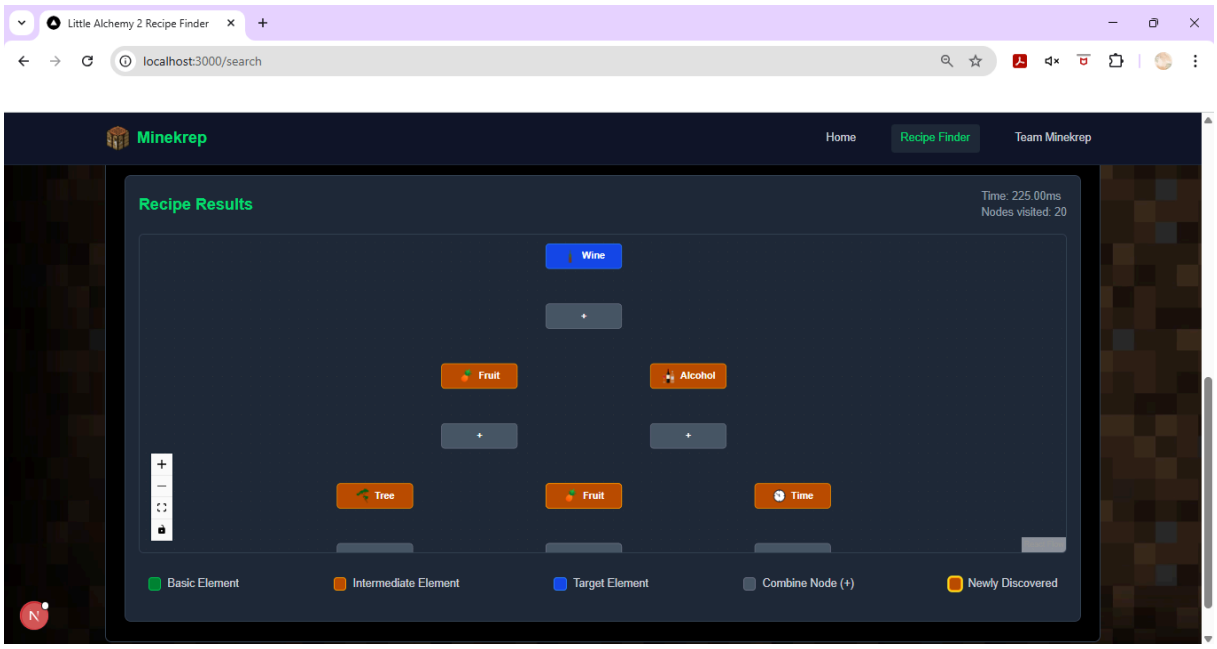
Gambar 4. *Landing Page*

4.6.2 Halaman Pencarian

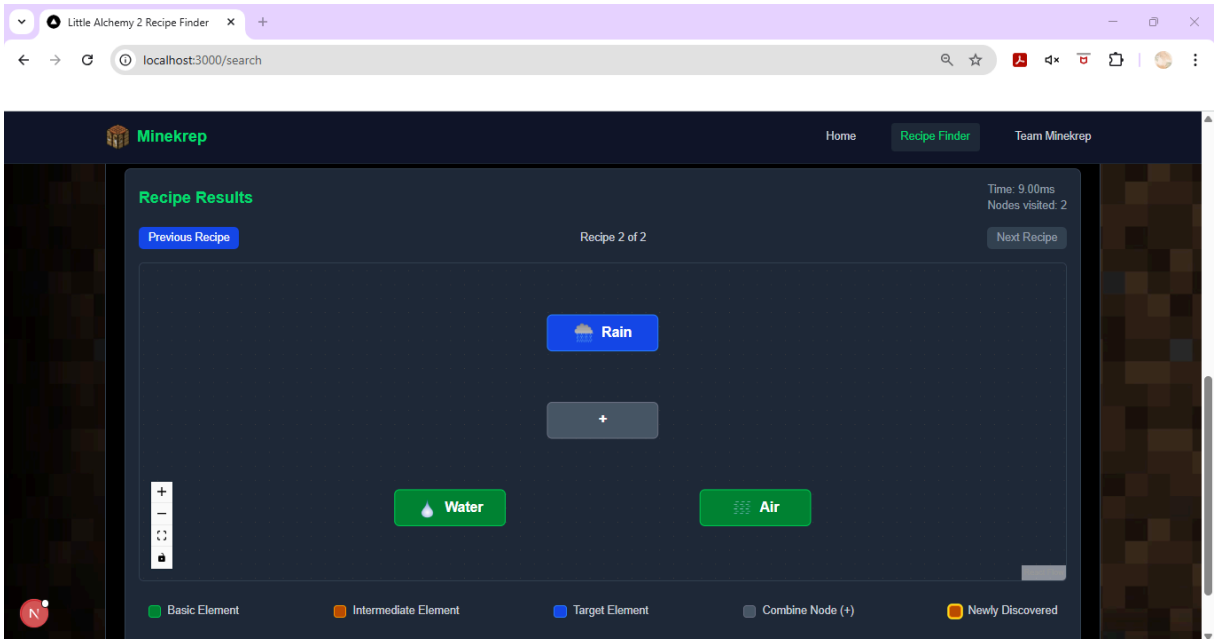
Pada halaman pencarian, pengguna dapat memasukkan nama elemen yang ingin dicari, jenis algoritma yang digunakan untuk pencarian, dan jumlah resep yang diinginkan. Pencarian akan dilakukan setelah pengguna menekan tombol "Find Recipes". Hasil pencarian ditampilkan di kolom "Recipe Results" yang bisa dinavigasi dengan *next recipe* serta *previous recipe* apabila melakukan pencarian *multiple recipe*.



Gambar 5. Halaman Pencarian Bagian Masukan Pengguna



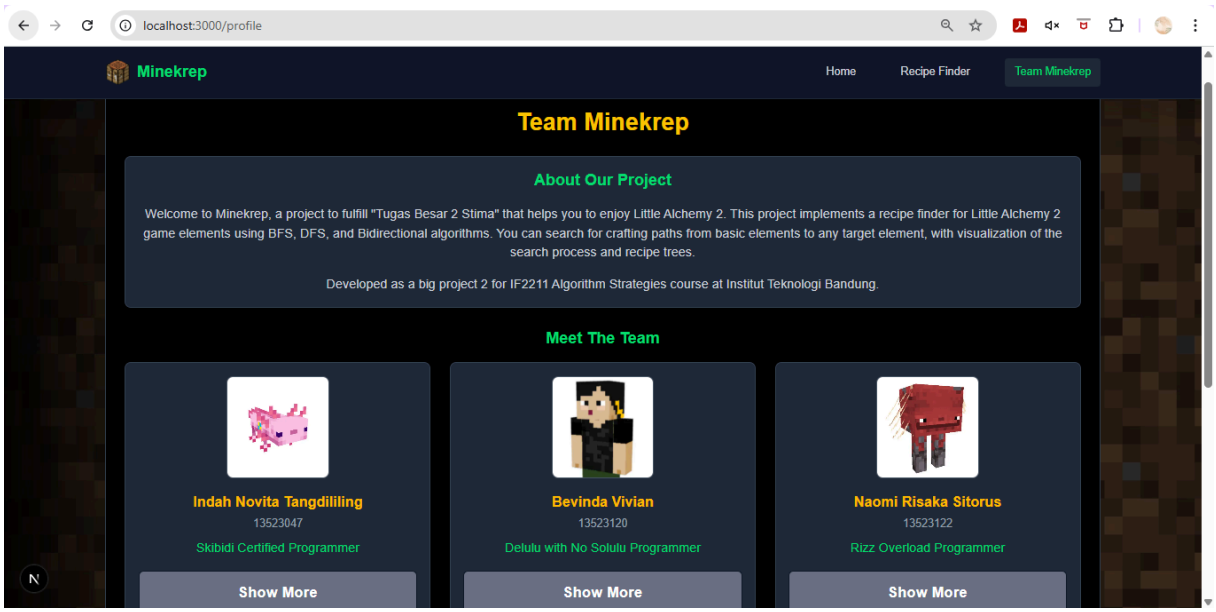
Gambar 6. Halaman Pencarian Bagian Hasil *Single Recipe*



Gambar 7. Halaman Pencarian Bagian Hasil *Multiple Recipe*

4.6.3 Halaman Profil

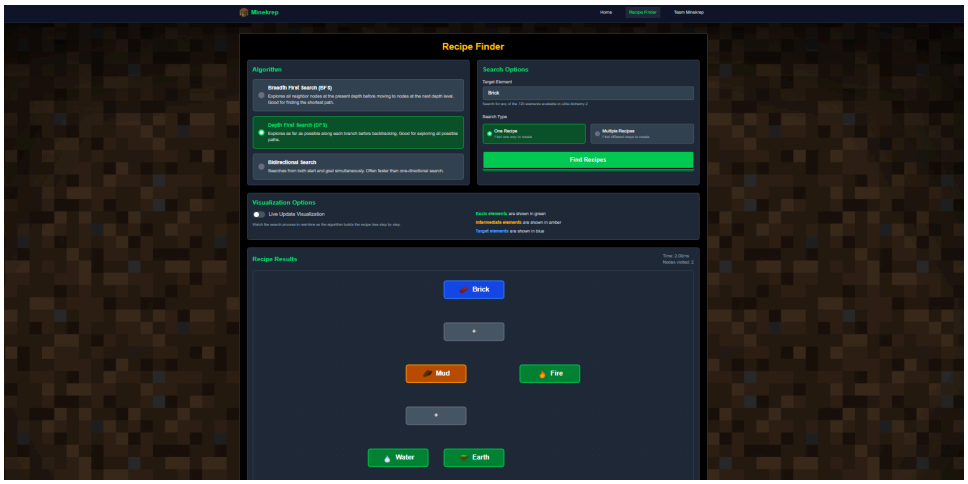
Halaman profil berisi informasi mengenai proyek, seperti teknologi yang digunakan, serta anggota kelompok Minekrep.



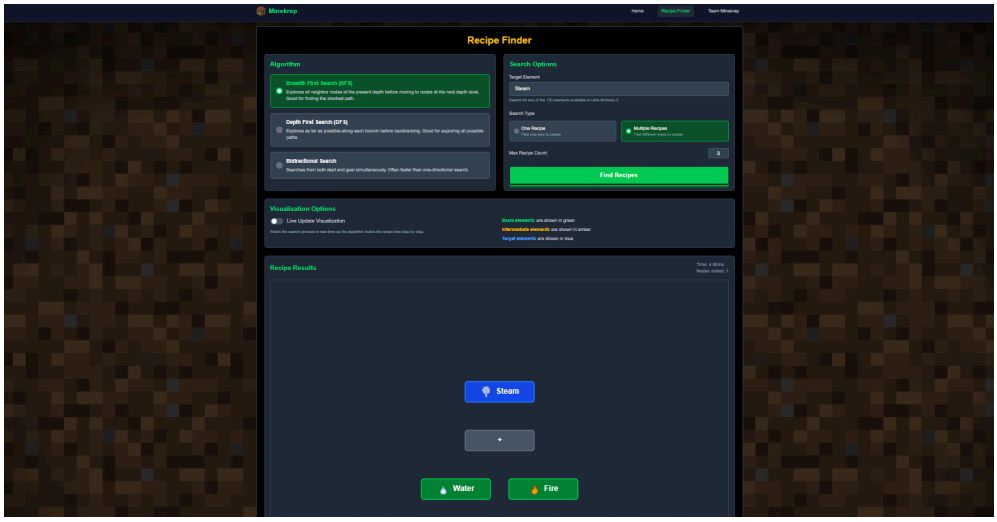
Gambar 8. Halaman Profil

4.7 Pengujian

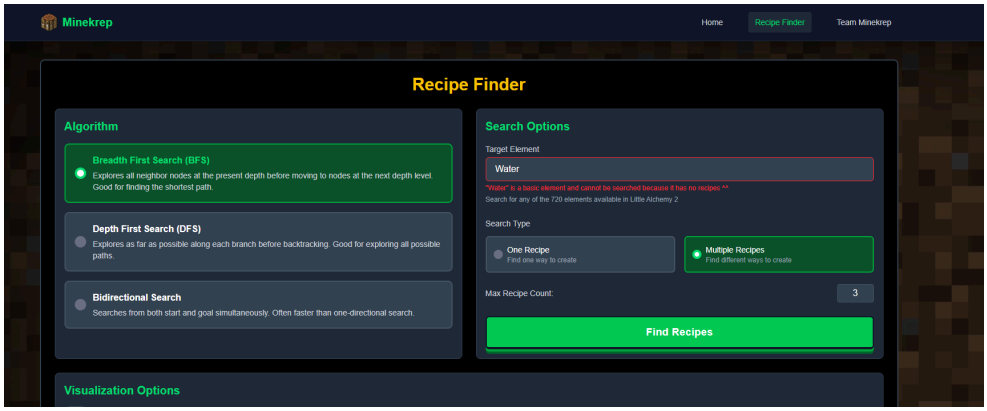
Gambar 9. Hasil Pencarian "Cashmere"



Gambar 10. Hasil Pencarian "Brick"



Gambar 11. Hasil Pencarian "Steam"



Gambar 12. Hasil Pencarian "Water"

4.8 Analisis Hasil Pengujian

4.8.1 Cashmere

Pengujian dilakukan dengan input target "Cashmere" algoritma BFS dan maksimum resep 3 buah. Seperti yang dapat dilihat pada Gambar 9, ada tombol "Next Recipe" dan "Previous Recipe" yang menandakan ada beberapa resep hasil yang dapat dilihat.

4.8.2 Brick

Pengujian dilakukan dengan input target “Brick” algoritma DFS dan “single recipe” yang berarti hanya mencari satu buah resep. Seperti yang dapat dilihat pada Gambar 10, program mengunjungi 5 nodes. Program juga menghasilkan resep Brick yang valid karena tier bahan-bahannya berada di bawah tier target.

4.8.3 Steam

Pengujian menginput target “Steam” algoritma DFS dan “multiple recipe” dengan maksimal resep 3 buah. Seperti yang dapat dilihat pada Gambar 11, hanya mengembalikan satu resep yaitu Water + Fire. Hal ini terjadi karena pada data yang dimiliki, hanya ada satu resep valid yang tidak melanggar aturan tier. Maka, program hanya menampilkan satu resep tersebut.

4.8.4 Water

Pengujian menginput target “Water”. Seperti yang dapat dilihat pada Gambar 12, pencarian tidak dilakukan. Diberikan juga catatan error berwarna merah di bawah input: ‘Water’ is a basic element and cannot be searched because it has no recipes ^^. Ini merupakan contoh handle kasus saat pengguna berusaha mencari resep dari basic element.

Bab 5

Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Dalam Tugas Besar 2 ini, kami berhasil membuat *website* pencari resep untuk permainan Little Alchemy 2 dengan *framework frontend* Next.js dan React serta bahasa CSS, Javascript, dan JSX, serta *backend* dengan bahasa Go. Pencarian resep mengimplementasikan algoritma BFS (Breadth First Search) dan DFS (Depth First Search) dengan jumlah resep yang dicari bisa satu atau lebih. Setiap elemen diwakilkan sebagai sebuah simpul dan resep yang menggabungkan dua elemen sebagai sisi dalam pembentukan graf, Resep itu sendiri kami olah dalam format JSON sebagai hasil dari *web scraping* secara statis. Kami juga mengimplementasikan *multithreading* pada pencarian *multiple recipes* demi meningkatkan efisiensi waktu program.

5.2 Saran

Saran dari kelompok Minekrep untuk tugas besar selanjutnya adalah perlunya spesifikasi yang lebih mendetail serta menggunakan bahasa yang tidak ambigu. Kebingungan mengenai spesifikasi membuat kelompok harus berasumsi sehingga memungkinkan terjadinya kesalahpahaman. Masih terdapat pula ketidakkonsistenan jawaban dari form QnA sehingga pemahaman akan spesifikasi menjadi ambigu.

5.3 Refleksi

Melalui tugas besar ini, kelompok telah belajar banyak mengenai konsep BFS serta DFS dan pengaplikasiannya dalam kasus nyata. Selain itu, tugas besar ini juga mendorong kelompok untuk melakukan eksplorasi lebih lanjut dalam hal *web development*, baik dari segi *frontend* maupun *backend*. Tugas besar ini juga memberikan kami kesempatan untuk mempelajari bahasa Go yang sedang populer digunakan untuk pertama kalinya. Walaupun terdapat tantangan dalam mengerjakan tugas besar ini, terutama dari segi waktu akibat banyaknya tugas

yang berjalan secara bersamaan, kelompok telah berusaha untuk memberikan yang terbaik dan mendapatkan pelajaran yang berharga dari tugas ini.

Lampiran

Pranala *repository* :

https://github.com/naomirisaka/Tubes2_Minekrep

Pranala website deploy:

https://minekrep-2025.vercel.app/

Pranala video pada Youtube:

https://youtu.be/XzAc8ymZ3Lw?si=Ye6Y4w4475eYkFu8

Tabel keterselesaian:

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	v	
2	Aplikasi dapat memperoleh data *recipe* melalui scraping.	v	
3	Algoritma *Depth First Search* dan *Breadth First Search* dapat menemukan *recipe* elemen dengan benar.	v	
4	Aplikasi dapat menampilkan visualisasi *recipe* elemen yang dicari sesuai dengan spesifikasi.	v	
5	Aplikasi mengimplementasikan multithreading.	v	
6	Membuat laporan sesuai dengan spesifikasi.	v	
7	Membuat bonus video dan diunggah pada Youtube.	v	

8	Membuat bonus algoritma pencarian *Bidirectional*.	v	
9	Membuat bonus *Live Update*.	v	
10	Aplikasi di-*containerize* dengan Docker.	v	
11	Aplikasi di-*deploy* dan dapat diakses melalui internet.	v	

Daftar Pustaka

Munir, R. (2025). Breadth First Search (BFS) dan Depth First Search (DFS) – Bagian 1. Diakses pada 08 Mei 2025 pukul 19.30 WIB melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf

Munir, R. (2025). Breadth First Search (BFS) dan Depth First Search (DFS) – Bagian 2. Diakses pada 08 Mei 2025 pukul 20.00 WIB melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf