Tugas Besar 2 IF2211 Strategi Algoritma
Semester II tahun 2024/2025

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

Disusun oleh:

Kelompok 61 WebBrewery

Muhammad Aufa Farabi	13523023
Darrel Adinarya Sunanda	13523038
Aryo Wisanggeni	13523100

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Gambar 1. Little Alchemy 2
(sumber: https://www.thegamer.com)

Little Alchemy 2 merupakan permainan berbasis web / aplikasi yang dikembangkan oleh Reclock 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. Energi
   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

1. Dasar Teori

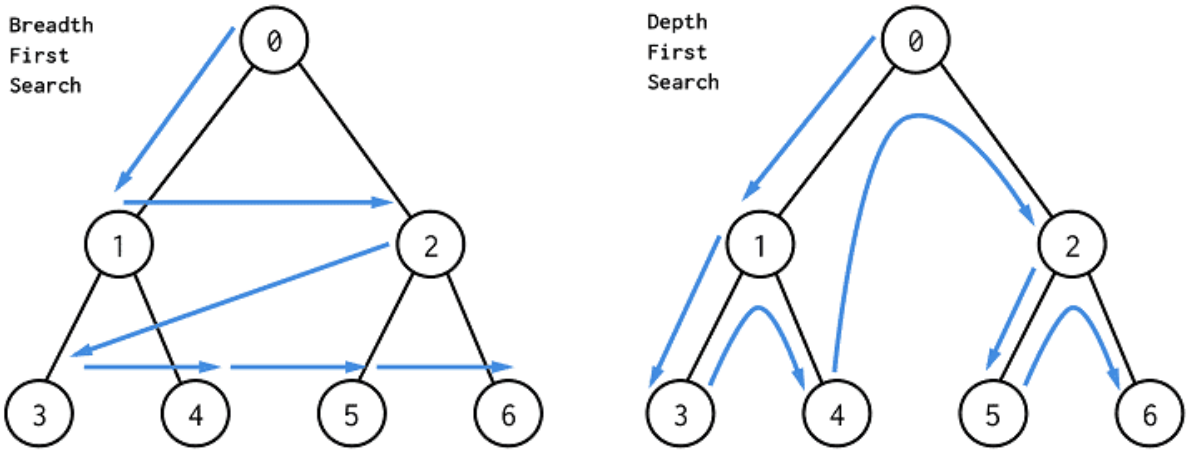
Traversal graf adalah proses mengunjungi simpul-simpul dalam sebuah graf secara sistematis untuk mencari solusi dari permasalahan yang direpresentasikan oleh graf tersebut. Dua metode traversal yang umum digunakan adalah Breadth-First Search (BFS) dan Depth-First Search (DFS), dengan asumsi graf yang digunakan terhubung. Berdasarkan ketersediaan informasi, algoritma pencarian dapat diklasifikasikan menjadi pencarian tanpa informasi (uninformed search) seperti BFS dan DFS, serta pencarian dengan informasi (informed search) seperti A*. Selain itu, graf dalam proses pencarian dapat direpresentasikan secara statis, yaitu sudah tersedia sejak awal, atau secara dinamis, yakni graf dibentuk selama pencarian berlangsung.

a. Breadth First Search (BFS)

   BFS merupakan algoritma penelusuran graf secara menyeluruh dengan strategi lebar-berdasarkan-level. BFS memulai penelusuran dari suatu simpul awal, kemudian mengunjungi semua simpul tetangganya sebelum melanjutkan ke simpul-simpul pada tingkat berikutnya. Implementasi BFS biasanya menggunakan struktur data *queue* untuk menjaga urutan penelusuran simpul.

b. Depth First Search (DFS)

   DFS adalah algoritma penelusuran yang menjelajahi simpul-simpul graf dengan strategi sedalam mungkin sebelum kembali (*backtrack*) dan menelusuri jalur lainnya. DFS bisa diimplementasikan menggunakan rekursi atau struktur data *stack*.



Gambar 2.1 Ilustrasi Perbedaan BFS dan DFS

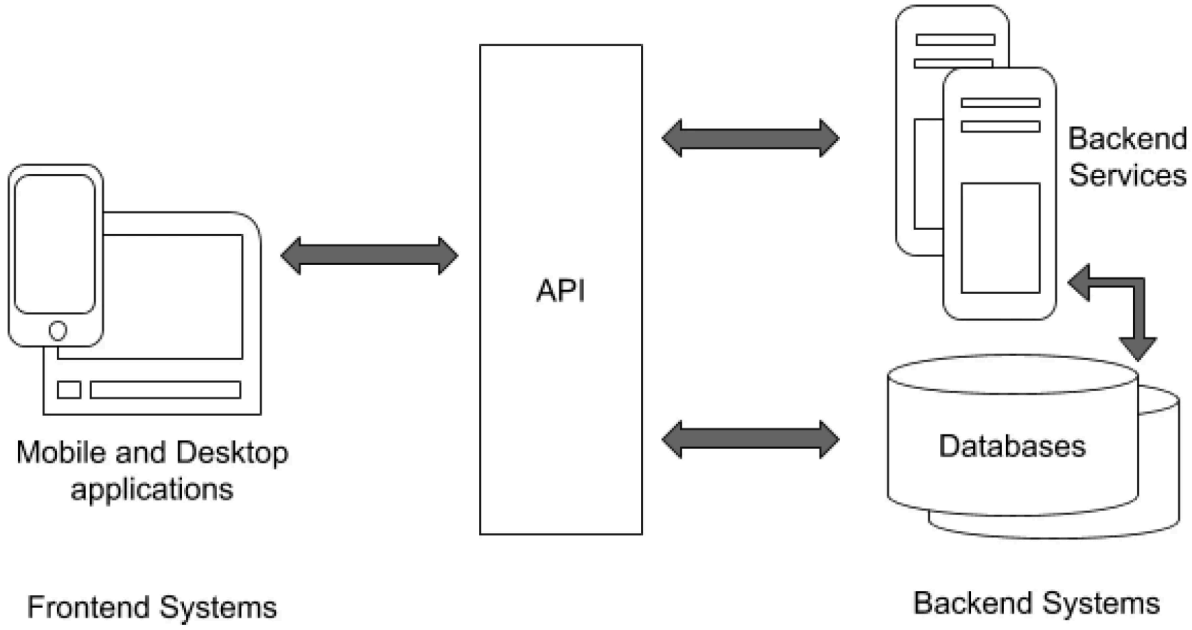
(Sumber: What is the Difference Between BFS and DFS Algorithms - Developers, Designers & Freelancers - FreelancingGig)

2. Aplikasi Web

Aplikasi web modern umumnya dibangun dengan arsitektur yang memisahkan antara frontend, backend, dan API sebagai penghubungnya. Pendekatan ini memungkinkan pengembangan sistem yang lebih modular dan fleksibel, serta mempermudah pemeliharaan dan pengembangan lanjutan.

Frontend adalah bagian dari aplikasi yang langsung berinteraksi dengan pengguna. Ia bertanggung jawab menampilkan antarmuka grafis, menerima input pengguna, dan menyajikan hasil secara visual. Di sisi lain, backend menangani logika utama aplikasi, termasuk pemrosesan algoritma, pengelolaan data, dan pengambilan keputusan berdasarkan permintaan yang diterima.

Untuk menghubungkan keduanya, digunakan API (Application Programming Interface) yang berfungsi sebagai perantara komunikasi antara frontend dan backend. API menerima permintaan dari frontend, memprosesnya di backend, lalu mengembalikan hasil yang kemudian ditampilkan kembali ke pengguna.



Gambar 2.2 The Role of APIs

(Sumber: A Seven-Step Guide to API-First Integration - InfoQ)

BAB III
ANALISIS PEMECAHAN MASALAH

1. Langkah-langkah Pemecahan Masalah

Masalah yang ingin dipecahkan adalah penemuan semua kemungkinan resep untuk membuat suatu elemen. Dengan demikian, alur untuk menemukan resep dari suatu elemen dapat dilakukan sebagai berikut:

1. Perhatikan bahwa setiap elemen dibuat dari dua elemen lainnya.
2. Terdapat elemen dasar, yaitu elemen yang tidak dibuat dari elemen lain.
3. Untuk menemukan resep suatu elemen, kita dapat menelusuri elemen-elemen pembentuknya secara rekursif, hingga mencapai elemen dasar.
4. Resep lengkap dari suatu elemen adalah sebuah pohon dengan akar elemen tersebut dan setiap daunnya berupa sebuah elemen dasar.

2. Proses pemetaan masalah menjadi elemen algoritma DFS & BFS

Pada masalah pencarian recipe untuk suatu elemen, untuk BFS dan DFS, simpul untuk membuat graf pencarian dinamis adalah satu elemen dan satu set recipe. Satu set recipe yang dimaksud adalah dua elemen lain yang memiliki tier yang lebih rendah dari elemen produk. Dengan demikian, berikut adalah elemen-elemen DFS dan BFS dari masalah ini.

1. Operator: tambah simpul (produk dan satu set recipe)
2. Akar: elemen target
3. Simpul: elemen produk dan satu set recipe
4. Daun: semua elemen pada suatu path didekomposisi menjadi elemen dasar (Fire, Water, Air, Earth, Time)
5. Ruang solusi: semua path dari elemen target yang semua elemen pembuatnya dapat didekomposisi menjadi elemen dasar
6. Ruang status: semua simpul (elemen produk dan satu set recipe) yang ada pada suatu saat pada pohon

3. Fitur fungsional dan arsitektur aplikasi web

Fitur fungsional dari aplikasi web yang dibangun adalah sebagai berikut:

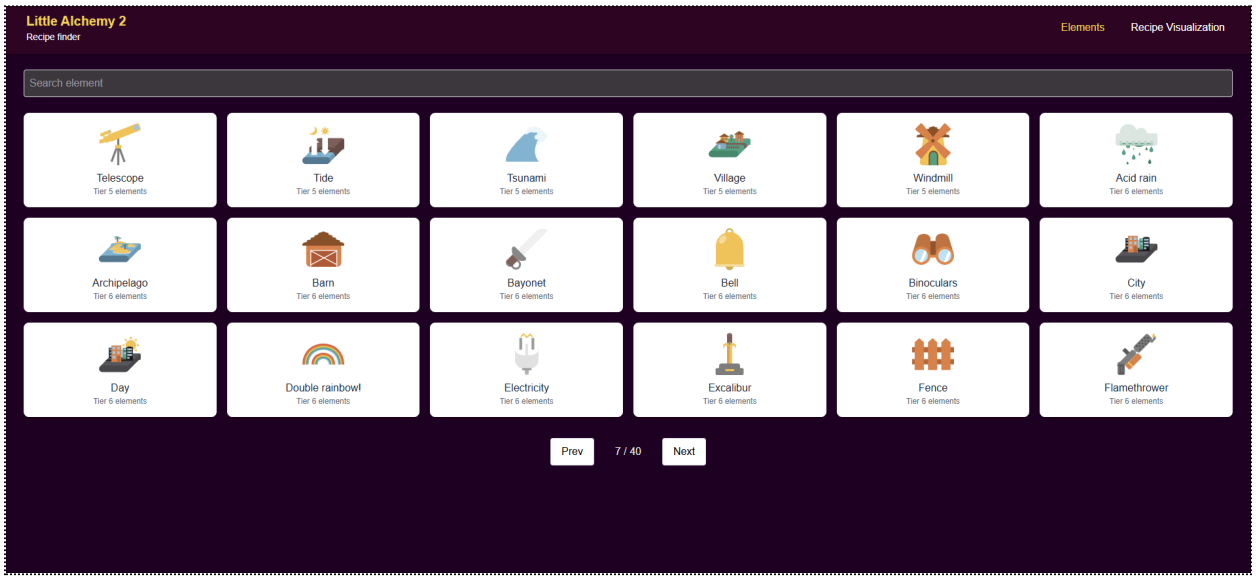
1. Tampilan semua elemen Little Alchemy beserta recipe yang dimiliki
2. Search bar untuk mencari elemen yang ingin divisualisasikan pohon resepnya
3. Visualisasi pohon resep elemen dengan rute recipe terpendek
4. Visualisasi pohon-pohon resep element sebanyak recipe yang dimiliki elemen
5. Mode pembangunan visualisasi pohon recipe dengan BFS
6. Mode pembangunan visualisasi pohon recipe dengan DFS
7. Mode pembangunan visualisasi pohon recipe secara Bidirectional (BFS dan DFS)

Arsitektur aplikasi web dibagi menjadi dua bagian, yaitu Frontend dan Backend. Pengguna mengakses Frontend untuk melihat daftar elemen yang tersedia beserta recipe yang dimiliki setiap elemen. Selain itu, pada frontend, pengguna dapat menyeleksi suatu elemen tertentu kemudian men-*toggle* algoritma pencarian dan banyak recipe yang diinginkan untuk menampilkan visualisasi pohon recipe. Data masukan dari pengguna tersebut dikirim ke backend untuk diproses yang diterima oleh API. Dari situ, API akan melakukan pencarian resep dengan algoritma yang dipilih pengguna. Hasil dari pencarian recipe oleh backend kemudian dikirimkan melalui response API yang diterima oleh frontend. Di sini, frontend akan membangun visualisasi pohon resep dan menampilkannya berdasarkan hasil recipe yang didapatkan oleh response API.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

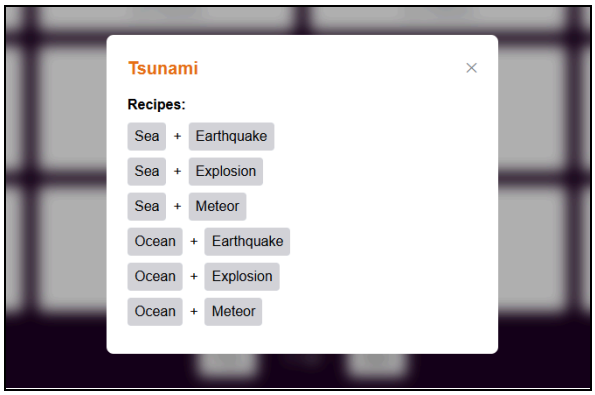
Frontend

Pada sisi frontend, aplikasi dibangun menggunakan framework Next.js yang merupakan pengembangan dari React dengan fitur tambahan seperti server-side rendering dan routing bawaan. Next.js dipilih karena kemampuannya dalam membangun antarmuka pengguna yang responsif dan efisien, serta kemudahan dalam mengelola struktur halaman dan komponen. Dalam aplikasi ini, Next.js digunakan untuk membuat halaman interaktif yang memungkinkan pengguna memasukkan input graf, memilih metode pencarian (BFS atau DFS), dan melihat hasil visualisasi jalur. Komponen React digunakan untuk menangani input dan tampilan, sementara komunikasi dengan backend dilakukan menggunakan HTTP request melalui API.



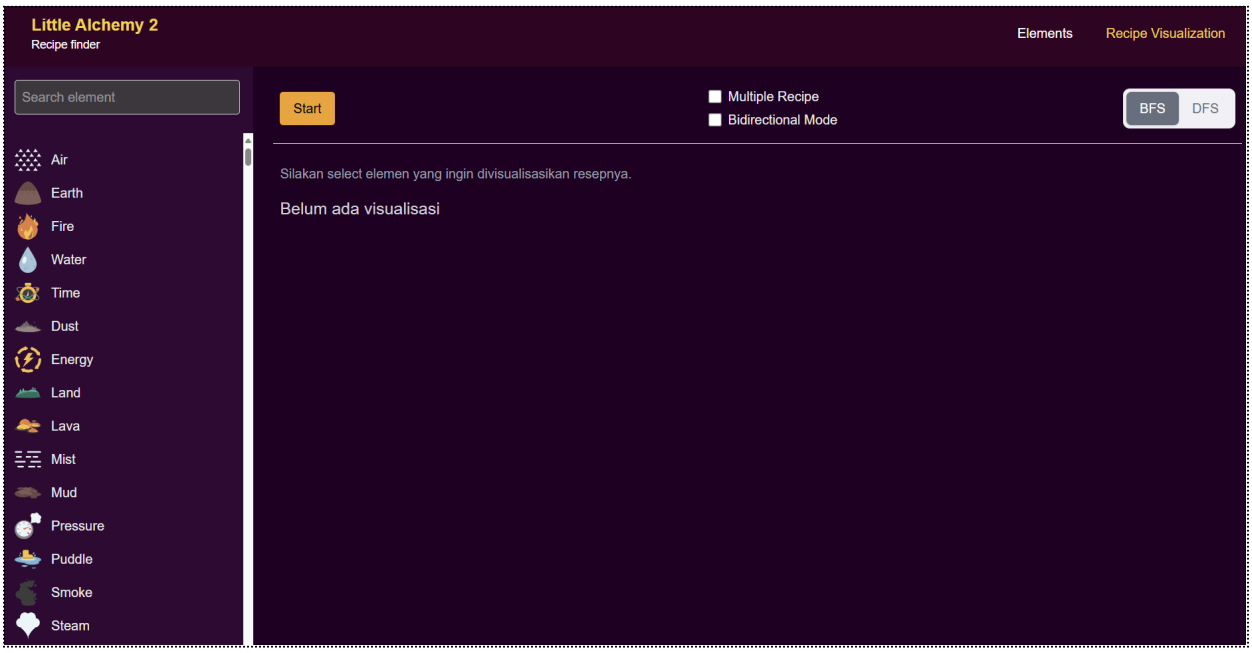
Gambar 4.1 Index Page

Antarmuka utama dari aplikasi menampilkan elemen-elemen. Di bagian atas terdapat kolom pencarian untuk memudahkan pengguna mencari elemen tertentu. Di bawahnya, elemen-elemen ditampilkan dalam bentuk kartu yang berisi ikon, nama elemen, dan kategori tier-nya. Navigasi antar halaman elemen disediakan melalui tombol “Prev” dan “Next” di bagian bawah. Pada bagian atas kanan halaman, terdapat menu navigasi untuk berpindah antar halaman “Elements” sekarang dan “Recipe Visualization”, di mana pengguna dapat melihat visualisasi resep dalam bentuk graf yang menunjukkan jalur pembuatan suatu elemen dari kombinasi elemen-elemen lainnya.



Gambar 4.2 Element Recipes

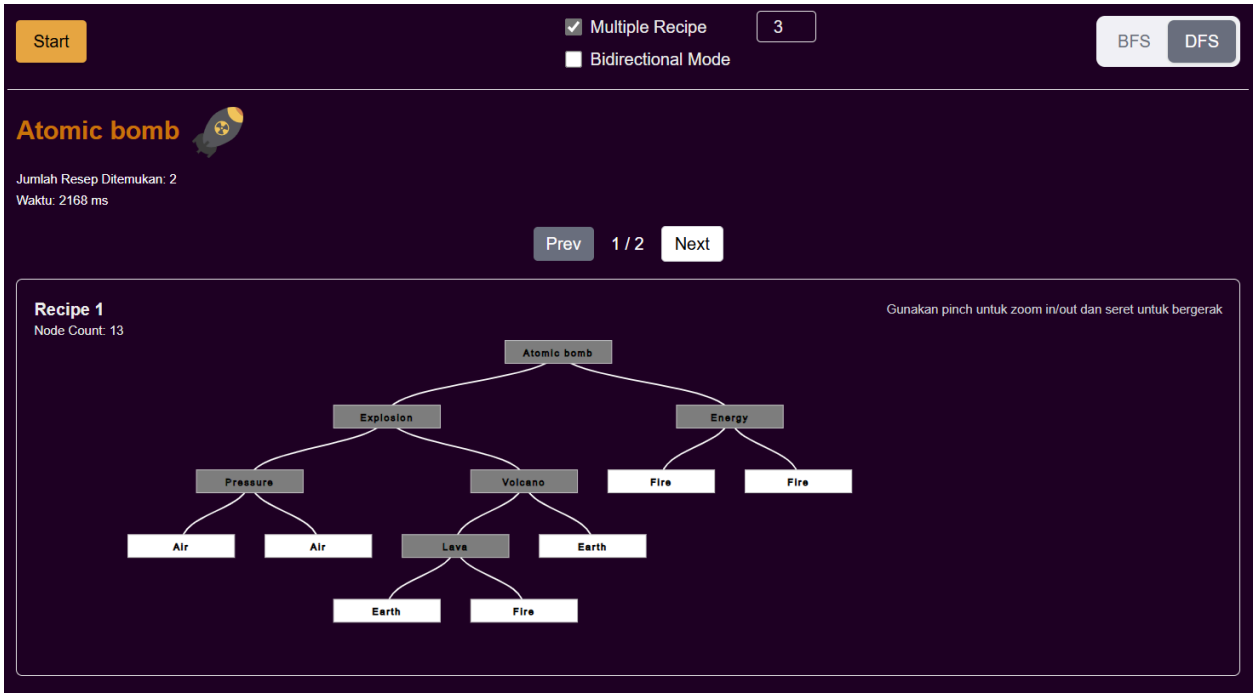
Ketika pengguna memilih salah satu elemen, aplikasi akan menampilkan jendela pop-up yang berisi semua kombinasi resep yang memungkinkan untuk menghasilkan elemen tersebut. Setiap resep ditampilkan dalam format dua elemen yang digabungkan, seperti "Sea + Earthquake" atau "Ocean + Meteor". Tampilan ini mempermudah pengguna dalam memahami berbagai cara untuk membuat elemen yang diinginkan, sekaligus memberikan fleksibilitas dalam eksplorasi kombinasi yang tersedia dalam permainan.



Gambar 4.3 Recipe Visualization

Halaman Recipe Visualization menampilkan antarmuka yang terbagi menjadi dua bagian utama. Di sisi kiri, terdapat panel yang memuat daftar elemen yang dapat ditelusuri, lengkap dengan ikon yang memudahkan identifikasi. Panel ini juga dilengkapi kolom pencarian di bagian atas untuk mempermudah pengguna dalam menemukan elemen tertentu. Di sisi kanan, terdapat area visualisasi resep yang akan menampilkan jalur pembuatan suatu elemen. Di bagian atas area ini, tersedia tombol Start untuk memulai proses visualisasi, serta dua opsi mode tambahan yaitu Multiple Recipe dan Bidirectional Mode yang dapat diaktifkan sesuai kebutuhan.

Pengguna juga dapat memilih algoritma pencarian yang digunakan, antara BFS (Breadth-First Search) atau DFS (Depth-First Search). Ketika belum ada elemen yang dipilih, area visualisasi akan menampilkan pesan “Belum ada visualisasi” yang menandakan bahwa pengguna perlu memilih elemen terlebih dahulu untuk melihat hasil visualisasi jalur pembuatannya.



Gambar 4.4 Contoh Recipe Visualization

Ketika pengguna memilih elemen (seperti Atomic bomb) pada halaman Recipe Visualization, sistem akan menampilkan detail resep pembuatannya secara visual dalam bentuk diagram pohon. Di bagian atas, ditampilkan nama elemen, ikon, jumlah resep yang ditemukan, serta waktu pencarian (dalam mikrosekon). Visualisasi memperlihatkan bahwa Atomic bomb berasal dari kombinasi Explosion dan Energy yang sendirinya dibuat dari kombinasi elemen-elemen lainnya.

Setiap elemen digambarkan dalam kotak dan dihubungkan oleh garis, membentuk struktur hierarkis yang menjelaskan urutan pembentukan elemen secara bertahap. Diagram ini bersifat interaktif, memungkinkan pengguna untuk melakukan zoom dan drag untuk navigasi yang lebih mudah sehingga mempermudah pemahaman proses kombinasi elemen secara visual dalam permainan.

```
const urls = [
    'https://web-brewery-kitchen-957948630882.asia-southeast2.run.app/api/recipe',
    // I read that this shouldn't be hardcoded. Welp.
    'http://localhost:8080/api/recipe'
];
// Falls back to a local backend if the online deployment fails

let response;
for (const url of urls) {
    try {
        response = await fetch(url, {
            method: 'POST',
            headers: {
                'Content-type': 'application/json',
            },
            body: JSON.stringify(payload),
        });
      ...
```

Gambar 4.5 API Call menuju Backend

Dalam implementasi aplikasi ini, proses pengambilan data atau resep dari backend dilakukan melalui pemanggilan API. Untuk memastikan aplikasi tetap dapat berjalan baik saat dikembangkan secara lokal maupun ketika sudah di-deploy ke server, digunakan dua alamat tujuan backend yang berbeda. Aplikasi pertama-tama mencoba untuk menghubungi server backend yang sudah di-deploy ke server cloud. Jika backend online ini tidak tersedia atau tidak merespons, maka aplikasi akan secara otomatis mencoba menghubungi backend yang berjalan di lokal.

Backend

Sisi backend dari program ini menggunakan bahasa pemrograman Go. Terdapat tiga fungsi utama dari backend proyek ini:

1. Scraping element, recipe, dan gambar dari link
2. Pemrosesan pencarian recipe suatu elemen menggunakan algoritma BFS, DFS, dan bidirectional
3. Penyedia API untuk request dari frontend untuk recipe menggunakan multithreading

Berikut adalah implementasi dari masing-masing fungsi tersebut.

A. Program Scraping Website Little Alchemy 2

Dari website Fandom Little Alchemy 2, diperoleh 720 elemen unik dengan berbagai tier, dimulai dari tier 0 (elemen dasar dan *time*) hingga elemen tier 15. Struktur untuk menyimpan hasil scraping website adalah JSON dengan struct Go sebagai berikut.

types.go

```
type Element struct {
      Name   string          `json:"name"`
      Category string      `json:"category"`
```

```
	Recipes  [][]string `json:"recipes"`
	Image string        `json:"image"`
}
```

Dengan struktur data tersebut, setiap elemen disimpan dengan format Name sebagai entri salah satu elemen, Category yang merupakan tier dari suatu elemen, Recipes berupa recipe-recipe yang dapat dikombinasikan untuk membuat suatu elemen, dan Image yang merupakan nama image dari setiap elemen beserta extensionnya, yaitu SVG.

B. Algoritma Pencarian Recipe

Untuk semua algoritma yang diterapkan, terdapat satu struktur node yang digunakan untuk membuat pohon dari recipe-recipe yang ada:

types.go

```
type Node struct {
	Name string
	Ingredient1 string
	Ingredient2 string
}
```

Struktur tersebut merupakan struktur yang lebih sederhana dari Elements dari Scrape.go. Struktur ini disederhanakan untuk Recipe yang diubah menjadi hanya Ingredient1 dan Ingredient2 karena sudah tahu pasti semua elemen di game ini dapat dibuat oleh 2 elemen lain, kecuali elemen dasar. Dengan struktur ini, ditambah dengan struktur list dari go, path dari setiap elemen dapat dibuat dan dipergunakan untuk membuat tree.

1. Algoritma BFS

Breadth First Search (BFS) yang diterapkan untuk memecahkan masalah ini terdapat dua pendekatan yang diterapkan: BFS dari elemen dasar dan BFS dari elemen target. Pendekatan pertama merupakan cara yang sangat boros secara memori dan juga lambat, ditambah lagi setelah menemukan target, algoritma untuk mencari elemen-elemen yang belum dikembangkan adalah pendekatan dua. Dari itu, untuk meningkatkan keefisienan, algoritma BFS yang digunakan adalah yang mulai langsung dari elemen target (yang dicari) dan mencari secara BFS hingga semua elemen pembuatnya bisa didekomposisi menjadi elemen tier 0 (elemen dasar). Satu heuristik yang juga digunakan pada pencarian ini adalah recipe dari suatu elemen tidak boleh lebih dari tier elemen itu sendiri. Heuristik ini ditambah untuk mencegah looping elemen A dan B, ketika A perlu B untuk dibuat dan B perlu A. Berikut adalah pseudocode dari algoritma ini.

bfs.go

```
func HeuristicalReverseBFS(target string, elements map[string]models.Element,
elementTier map[string]int, seed int) []models.Node {
   1. Cek apakah target adalah tier 0, jika iya, return
   2. Inisiasi queue BFS dengan elemen target beserta semua kombinasi
      recipenya yang mempunyai tier di bawah target
```

```
   3. Selama queue masih ada isi dan belum ada solusi, cari recipe-recipe
      untuk semua elemen yang belum selesai diekspansi hingga tier 0 dan
      kembangkan secara DFS
}
```

2. Algoritma DFS

Belajar dari eksperimen BFS, DFS langsung menggunakan algoritma yang dimulai dari target element. Untuk DFS, caranya cukup mudah, pilih salah satu recipenya yang valid dan kembangkan terus hingga mencapai element tier 0. Berikut adalah pseudocode singkat dari kode yang diimplementasi.

dfs.go

```
func ReverseDFS(target string, elements map[string]models.Element,
elementTier map[string]int, seed int, strict bool) []models.Node{
   1. Cek apakah target adalah tier 0, jika iya, return
   2. Pilih suatu elemen yang belum diekspansi dan panggil lagi ReverseDFS
      helper untuk elemen tersebut
   3. Ulangi (2) hingga di-output suatu path yang "penuh"
}
```

3. Algoritma BFS Bidirectional

Algoritma ini merupakan BFS yang dilakukan dari dua arah, dari arah elemen dasar dan dari arah elemen target. Untuk dari elemen dasar, ekspansi dilakukan secara asal selama ekspansi tidak melebihi tier elemen target, sedangkan dari elemen target, melakukan ekspansi sesuai dengan algoritma BFS normal. Dari kedua ekspansi tersebut, setiap langkah dicek apakah terjadi suatu pertemuan. Jika suatu pertemuan terjadi, ekspansi elemen-elemen yang belum dapat didekomposisi menjadi elemen dasar seperti pada BFS normal.

dfs.go

```
func BidirectionalBFS(target string, elements map[string]models.Element,
elementTier map[string]int, seed int) []models.Node{
   1. Inisiasi queue forward dan queue reverse
   2. Cek apakah terjadi pertemuan forward dan reverse queue
   3. Jika pertemuan terjadi ekspansi elemen yang belum didekomposisi jadi
      elemen dasar, return jika selesai
   4. Ekspansi forward queue secara BFS
   5. Ekspansi reverse queue secara BFS
   6. Kembali ke (2) jika kedua queue masih berisi
   7. Return jika kedua queue sudah kosong
}
```

4. Algoritma DFS Bidirectional

Algoritma ini merupakan DFS yang dilakukan dari dua arah, dari arah elemen dasar dan dari arah elemen target. Untuk dari elemen dasar, ekspansi dilakukan secara asal selama ekspansi tidak melebihi tier elemen target, sedangkan dari elemen target, melakukan ekspansi sesuai

dengan algoritma DFS normal. Dari kedua ekspansi tersebut, setiap langkah dicek apakah terjadi suatu pertemuan. Jika suatu pertemuan terjadi, ekspansi elemen-elemen yang belum dapat didekomposisi menjadi elemen dasar seperti pada DFS normal.

dfs.go

```
func BidirectionalDFS(target string, elements map[string]models.Element,
elementTier map[string]int, seed int) []models.Node{
   1. Inisiasi stack forward dan stack reverse
   2. Cek apakah terjadi pertemuan forward dan reverse stack
   3. Jika pertemuan terjadi ekspansi elemen yang belum didekomposisi jadi
      elemen dasar, return jika selesai
   4. Ekspansi forward stack secara DFS
   5. Ekspansi reverse stack secara DFS
   6. Kembali ke (2) jika kedua stack masih berisi
   7. Return jika kedua stack sudah kosong
}
```

C. API dengan Multithreading

Untuk API yang disediakan, struktur request dari frontend berupa sebagai berikut:

types.go

```
type RequestPayload struct {
      Target string      `json:"target"`
      Method string      `json:"method"`
      PathNumber  int          `json:"pathNumber"`
      Bidirectional bool       `json:"bidirectional"`
}
```

Struktur tersebut mempunyai 4 field. Pertama adalah target yang merupakan elemen yang ingin dicari oleh pengguna. Kedua adalah metode pencarian untuk mencari suatu recipe dari elemen target, terdapat hanya dua metode, yakni BFS dan DFS. Ketiga adalah PathNumber yang merupakan field untuk menyimpan jumlah maksimal path yang ingin ditampilkan oleh pengguna untuk suatu elemen target. Terakhir adalah field Bidirectional yang merupakan boolean untuk penambah metode pencarian BFS dan DFS.

Selain itu, terdapat juga struktur response dari API pada aplikasi ini. Struktur tersebut adalah sebagai berikut.

types.go

```
type ResponsePayload struct {
      Count int          `json:"count"`
      ElapsedTime int64        `json:"elapsedTime"`
      Paths []CountedPath `json:"paths"`
}
```

types.go

```
type CountedPath struct {
      NodeCount int      `json:"nodeCount"`
      Path   []Node      `json:"path"`
}
```

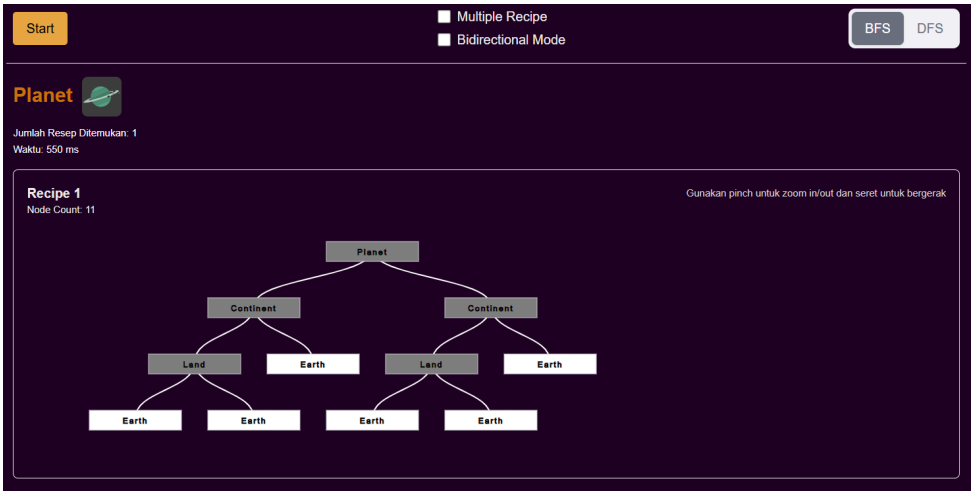
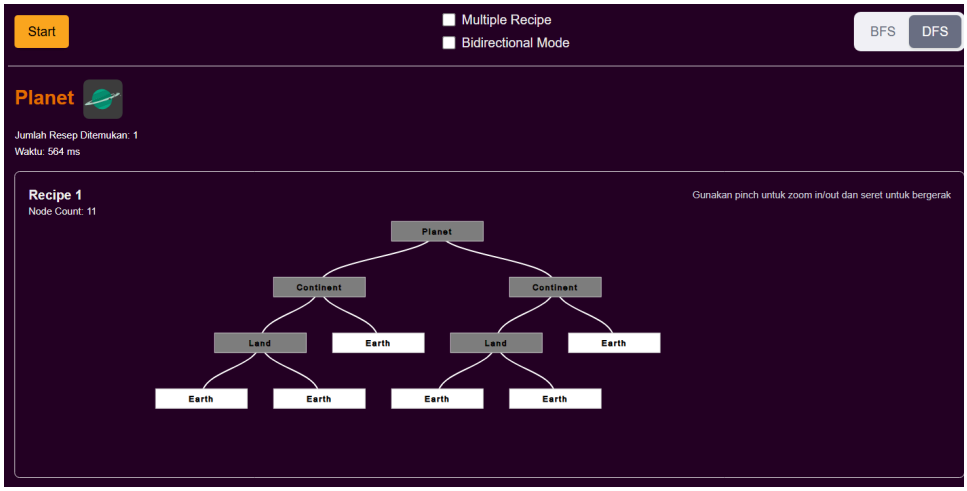
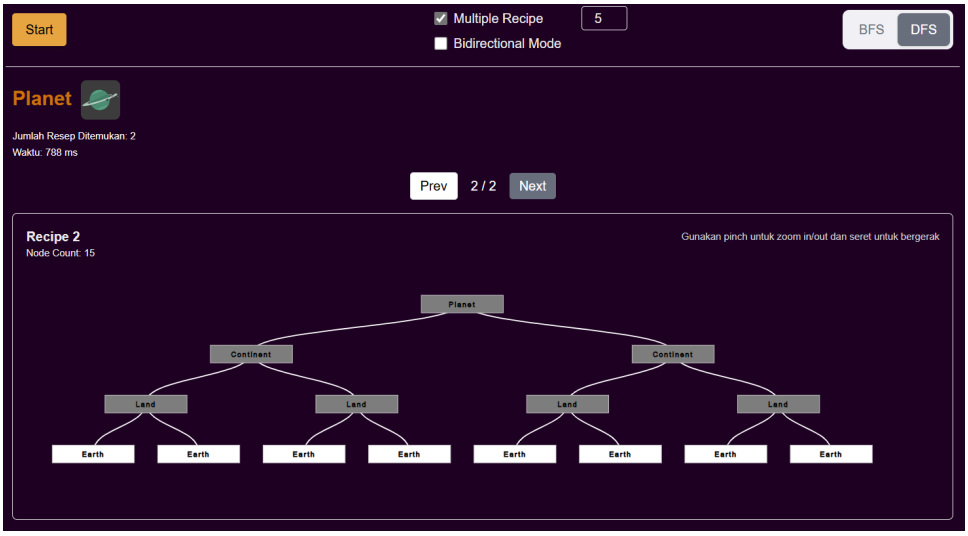
Terdapat dua struktur pada response, satunya merupakan elemen dari lainnya. Dimulai dengan struktur CountedPath, struktur ini merupakan representasi dari path recipe yang sudah lengkap beserta node count dari path tersebut. Selanjutnya untuk struktur ResponsePayload, struktur ini terdiri tiga field. Pertama adalah count, yaitu jumlah path yang diperoleh dari pencarian. Kedua adalah elapsed time yang merupakan waktu yang diperlukan untuk mencari path, untuk kasus multipath, elapsed time adalah kumulatif dari pencarian semuanya, namun karena dilakukan multithreading, waktu yang diberikan menjadi waktu yang dibutuhkan untuk menyelesaikan path yang paling lama diselesaikan. Terakhir adalah paths yang merupakan array dari CountedPath untuk menyimpan semua path yang diperoleh dengan jumlah sesuai dengan request pengguna.

Bagian terakhir dari API ini adalah fungsi multithreading yang digunakan untuk memproses semua path ketika request multipath secara bersamaan. Fungsi ini diimplementasi dengan pseudocode sebagai berikut.

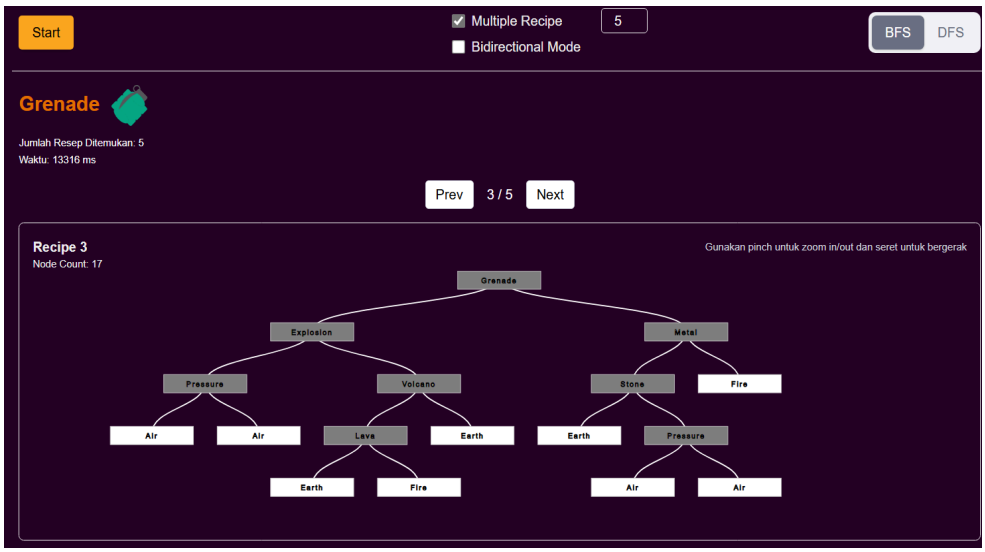
multithreading.go

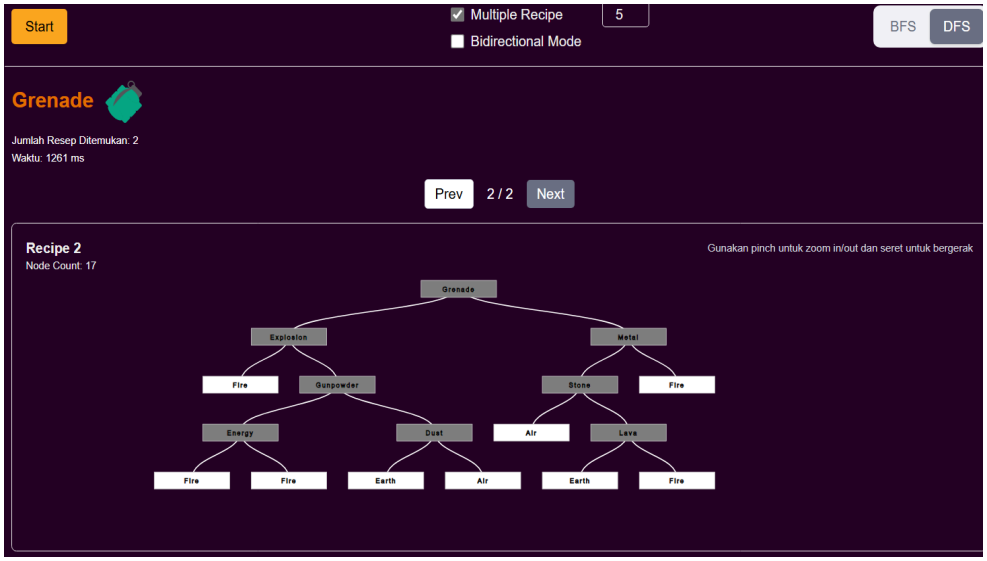
```
func ConcurrentPathFinder(pathNumber, maxIteration int, finder func(seed int)
[]models.Node, elementTier map[string]int) []models.CountedPath {
   1. Membuat buffer untuk menampung semua path yang akan dihasilkan
   2. Membuat buffer channel dengan semaphore untuk menghindar CPU thrashing
   3. Membuat goroutine untuk setiap seed disertai dengan WaitGroup
   4. Setiap pencarian dengan suatu seed dengan timer 500ms, jika pencarian
      melebihi waktu tersebut, akan dibuang
   5. Isi buffer awal dengan hasil dari pencarian yang tidak null dan membuang
      hasil duplikat
   6. Sortir hasil berdasarkan seed untuk kekonsistenan hasil
   7. Terakhir return buffer yang berisi solusi
}
```

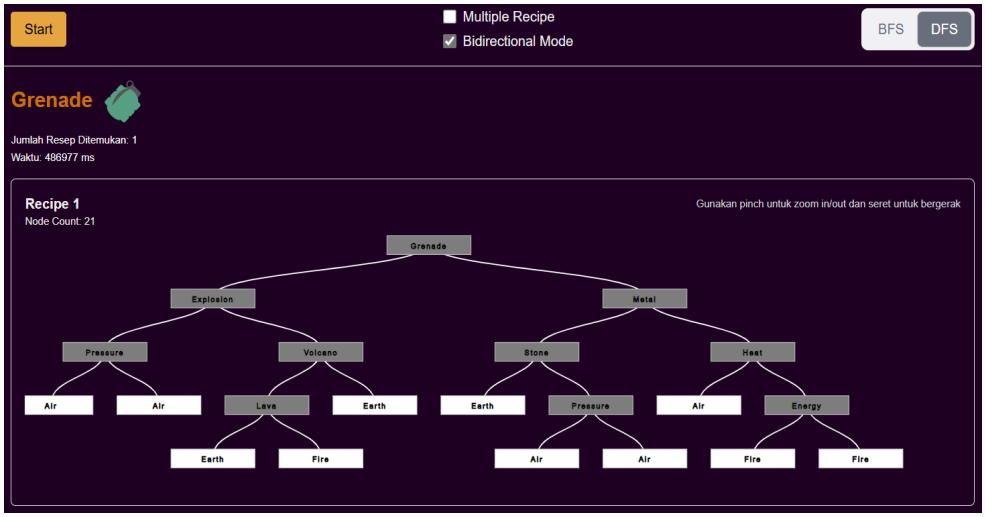
Pengujian

Pengujian	Hasil
(1) Planet	  

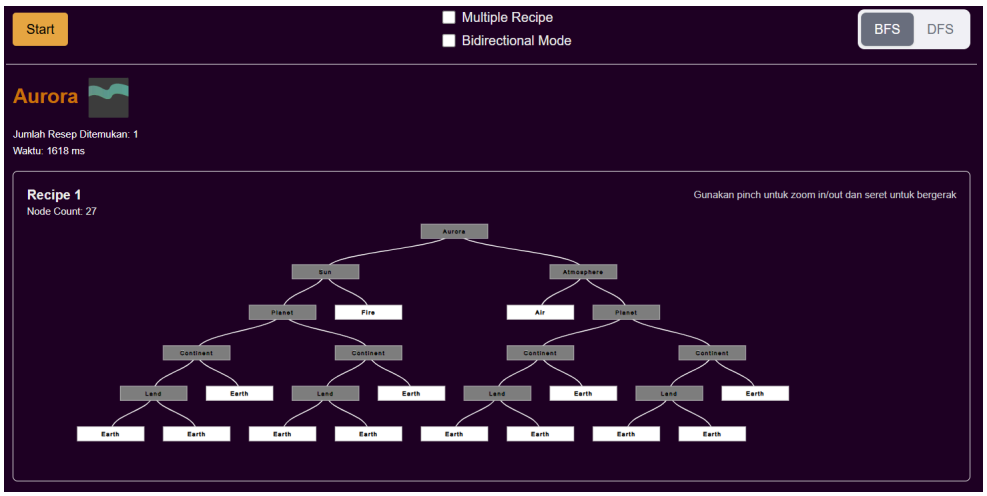
(2) Grenade

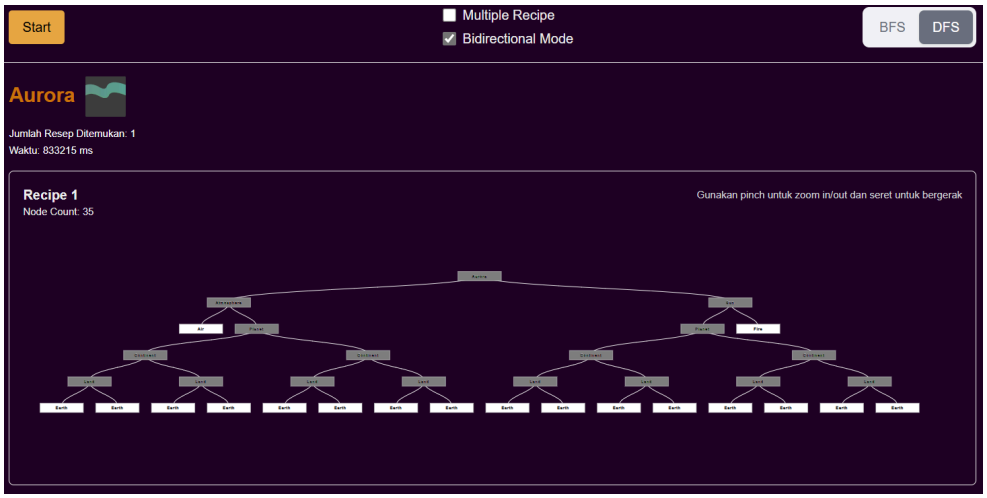

Start
Multiple Recipe
5
Bidirectional Mode
BFS
DFS
Grenade
Jumlah Resep Ditemukan: 5
Waktu: 13316 ms
Prev
3 / 5
Next
Recipe 3
Node Count: 17
Gunakan pinch untuk zoom in/out dan seret untuk bergerak
Grenade
Explosion
Metal
Pressure
Volcano
Stone
Fire
Air
Air
Lava
Earth
Earth
Pressure
Earth
Fire
Air
Air

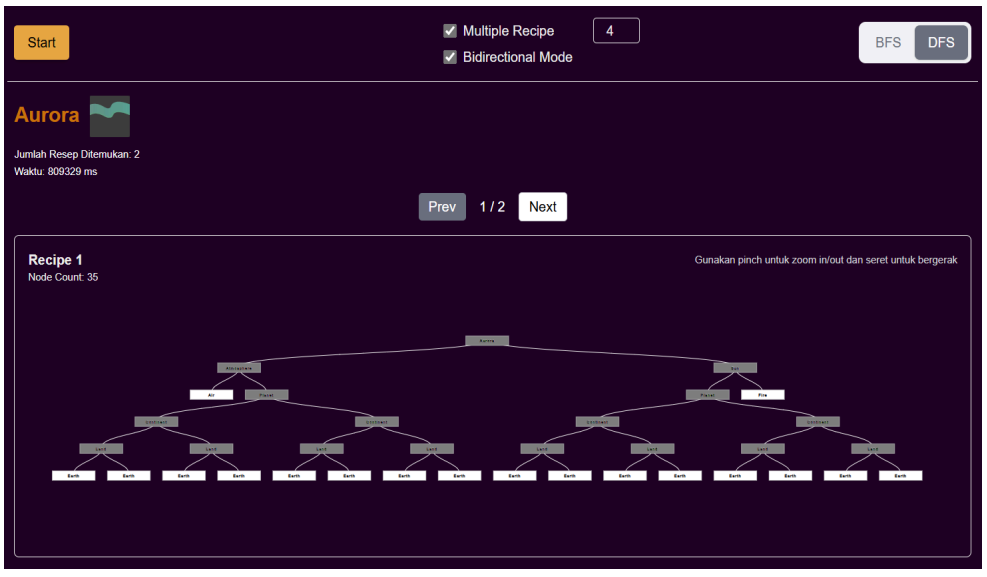

Start
Multiple Recipe
5
Bidirectional Mode
BFS
DFS
Grenade
Jumlah Resep Ditemukan: 2
Waktu: 1261 ms
Prev
2 / 2
Next
Recipe 2
Node Count: 17
Gunakan pinch untuk zoom in/out dan seret untuk bergerak
Grenade
Explosion
Metal
Fire
Gunpowder
Stone
Fire
Energy
Dust
Air
Lava
Fire
Fire
Earth
Air
Earth
Fire


Start
Multiple Recipe
Bidirectional Mode
BFS
DFS
Grenade
Jumlah Resep Ditemukan: 1
Waktu: 486977 ms
Recipe 1
Node Count: 21
Gunakan pinch untuk zoom in/out dan seret untuk bergerak
Grenade
Explosion
Metal
Pressure
Volcano
Stone
Heat
Air
Air
Lava
Earth
Earth
Pressure
Air
Energy
Earth
Fire
Air
Air
Fire
Fire

(3) Aurora


Start
Multiple Recipe
Bidirectional Mode
BFS
DFS
Aurora
Jumlah Resep Ditemukan: 1
Waktu: 1618 ms
Recipe 1
Node Count: 27
Gunakan pinch untuk zoom in/out dan seret untuk bergerak


Start
Multiple Recipe
Bidirectional Mode
BFS
DFS
Aurora
Jumlah Resep Ditemukan: 1
Waktu: 833215 ms
Recipe 1
Node Count: 35
Gunakan pinch untuk zoom in/out dan seret untuk bergerak


Start
Multiple Recipe
4
Bidirectional Mode
BFS
DFS
Aurora
Jumlah Resep Ditemukan: 2
Waktu: 809329 ms
Prev
1 / 2
Next
Recipe 1
Node Count: 35
Gunakan pinch untuk zoom in/out dan seret untuk bergerak

Analisis Pengujian

Dari hasil pengujian yang dilakukan, proses penelusuran resep untuk berbagai elemen berjalan dengan baik. Sistem mampu menampilkan langkah-langkah pembentukan elemen kompleks dalam bentuk pohon resep yang runtut dan mudah dipahami. Pada sampel kecil pengujian yang dilakukan, mode BFS cenderung lebih cepat dalam menemukan jalur terpendek menuju elemen target. Sebaliknya, mode DFS memberikan hasil yang lebih dalam dan mampu mengeksplorasi berbagai alternatif jalur.

Hal ini terlihat lebih dalam ketika fitur Multiple Recipe diaktifkan, ditemukan bahwa algoritma DFS cenderung menghasilkan lebih banyak variasi resep dibandingkan dengan BFS. Hal ini disebabkan oleh sifat dasar DFS yang menelusuri jalur pencarian hingga kedalaman maksimum sebelum mundur dan mencoba jalur lain. Dengan pendekatan ini, DFS mampu mengeksplorasi lebih banyak kombinasi elemen yang mungkin membentuk elemen target, terutama pada struktur pencarian yang memiliki banyak cabang atau kedalaman yang kompleks.

Sebaliknya, BFS fokus pada pencarian jalur terpendek dan akan lebih cepat menghentikan pencarian ketika elemen target ditemukan dalam level atas dari pohon pencarian. Akibatnya, meskipun BFS efisien dalam menemukan resep tercepat atau terpendek, jumlah variasi resep yang ditemukan cenderung lebih sedikit dibandingkan DFS. Hal ini terlihat dalam pengujian terhadap elemen-elemen seperti Grenade dan Aurora, di mana DFS mampu menemukan lebih banyak alternatif kombinasi bahan pembentuk dibandingkan BFS. Perbedaan ini menjadikan DFS lebih unggul dalam konteks eksplorasi menyeluruh ketika pengguna membutuhkan banyak opsi pembuatan elemen, sementara BFS lebih unggul dalam efisiensi waktu dan jalur minimal.

BAB V
KESIMPULAN, SARAN, DAN REFLEKSI

Kesimpulan

Pendekatan Breadth-First Search (BFS) maupun Depth-First Search (DFS) menunjukkan kinerja yang efektif dalam menemukan jalur pembuatan elemen. Kedua algoritma memiliki karakteristik dan keunggulan masing-masing dalam pencarian resep. BFS unggul dalam menemukan jalur tercepat dan paling efisien menuju elemen target, menjadikannya ideal untuk penelusuran yang mengutamakan efisiensi. Sementara itu, DFS lebih unggul dalam eksplorasi menyeluruh, terutama ketika fitur Multiple Recipe diaktifkan, karena mampu menemukan lebih banyak variasi kombinasi resep. Oleh karena itu, pemilihan algoritma penelusuran dapat disesuaikan dengan kebutuhan pengguna: BFS untuk efisiensi jalur, dan DFS untuk kelengkapan eksplorasi.

Saran

Untuk kedepannya, saran bagi kelompok kami adalah untuk meningkatkan komunikasi antar anggota lebih baik lagi, melakukan pengecekan lebih dalam atas setiap progres pengerjaan, dan lebih mendalami spek tugas beserta algoritma yang digunakan dalam program agar hasil program lebih optimal.

Refleksi

Refleksi yang didapatkan dari pengerjaan tugas ini adalah mengenai pentingnya komunikasi antar anggota dan perlunya pengecekan atas setiap sub-tugas yang dikerjakan, seperti testing fitur. Kami menyadari bahwa dalam pengerjaan tugas ini, masih ada kurangnya koordinasi dalam pengerjaan tugas dan program yang dibuat masih dapat dioptimalkan. Di sisi lain, kami juga belajar banyak mengenai pembuatan algoritma BFS dan DFS secara optimal pengalaman dalam mengembangkan aplikasi berbasis web.

LAMPIRAN

Pranala

Github Repository: https://github.com/Staryo40/Tubes2_WebBrewery
Aplikasi yang di-deploy: https://web-brewery-957948630882.asia-southeast2.run.app/

Checklist

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi dapat memperoleh data recipe melalui scraping.	✓	
3	Algoritma Depth First Search dan Breadth First Search dapat menemukan recipe elemen dengan benar.	✓	
4	Aplikasi dapat menampilkan visualisasi recipe 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 Bidirectional.	✓	
9	Membuat bonus Live Update.		✓
10	Aplikasi di-containerize dengan Docker.	✓	
11	Aplikasi di-deploy dan dapat diakses melalui internet.	✓	

DAFTAR PUSTAKA

Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 1
(Diakses pada tanggal 12 Mei 2025)

Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 2
(Diakses pada tanggal 12 Mei 2025)

Next.js Documentation
(Diakses pada tanggal 13 Mei 2025)

Documentation - The Go Programming Language
(Diakses pada tanggal 13 Mei 2025)

Docker Docs
(Diakses pada tanggal 13 Mei 2025)

Google Cloud Documentation
(Diakses pada tanggal 13 Mei 2025)