Group 10

-MONSTER CHASE-

MEET OUR MEMBERS

01 IZAAZ VERDIANSYAH KHAISAN ATHIF
- 5026241194

02 ARY RATNA AIDA SAFA
- 5026241029

03 GHANENDRA DZAKWAN WIRADIKUSUMAH
- 5026241127

04 CANDLELINE AUDRINA FIRSTA
- 5026221159

LATAR BELAKANG
&
RUMUSAN MASALAH

LATAR BELAKANG

Banyak mahasiswa mengalami kesulitan memahami materi Algoritma & Struktur Data karena penyajiannya cenderung abstrak, monoton, dan minim contoh interaktif sehingga konsep seperti graph, sorting, dan tree sering terasa sulit untuk dipahami. Untuk menjawab permasalahan tersebut, game labirin ini dikembangkan sebagai solusi pembelajaran yang lebih menarik, di mana algoritma BFS digunakan untuk pergerakan monster dan DFS digunakan untuk sistem Fog yang membatasi penglihatan pemain. Melalui gameplay yang menantang serta trap berisi pertanyaan materi ASD, game ini membantu mahasiswa belajar secara lebih aktif, kontekstual, dan menyenangkan tanpa terasa seperti pembelajaran formal.

RUMUSAN MASALAH

- Bagaimana membuat pembelajaran Algoritma & Struktur Data lebih menarik dan tidak monoton bagi mahasiswa?
- Bagaimana memberikan contoh penerapan nyata algoritma BFS dan DFS dalam sebuah sistem yang mudah dipahami?
- Bagaimana menghadirkan media latihan yang interaktif sehingga mahasiswa dapat mengulang materi ASD secara aktif dan tidak pasif?
- Bagaimana merancang game sederhana yang dapat menggabungkan konsep pathfinding (BFS), Fog (DFS), serta mekanisme trap untuk evaluasi pemahaman?

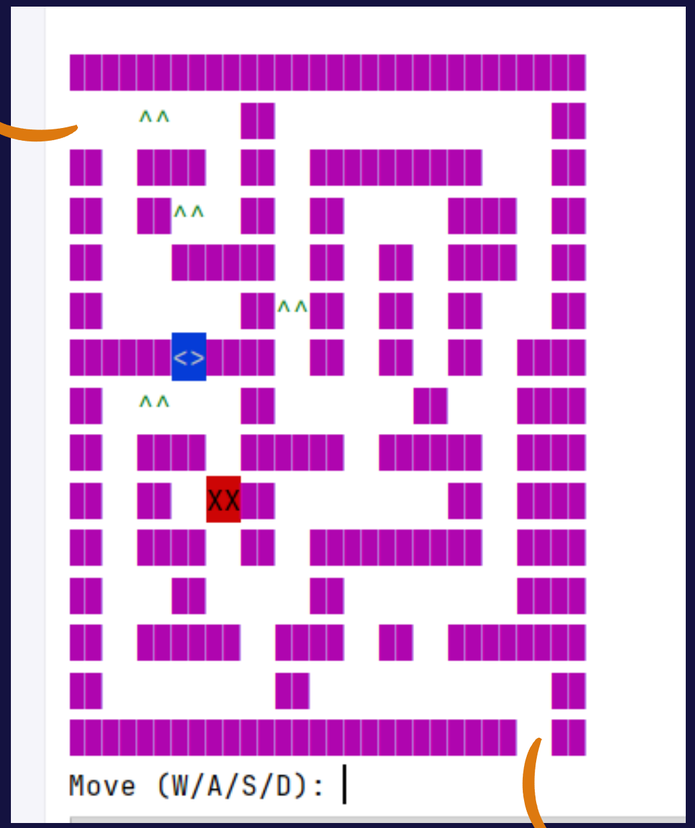
SOLUSI
-MONSTER CHASE-

Game yang menyediakan latihan soal Algoritma dan Struktur Data ke dalam gameplay yang interaktif dan menjadi alternatif latihan yang tidak monoton bagi mahasiswa.

START

DESKRIPSI-MONSTER CHASE

START



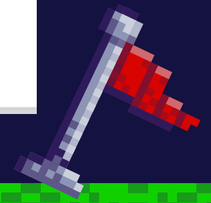
Monster Case adalah game turn-based player (bergerak 1 langkah per giliran). Ini adalah permianan eksplorasi labirin 15x15 di mana pemain harus keluar dengan menggunakan W A S D sambil menghindari monster.

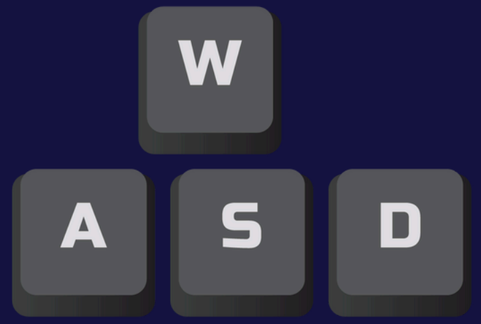
<> Karakter yang harus Anda gerakkan

XX Monster, Anda tidak boleh tertangkap oleh Monster ini!!

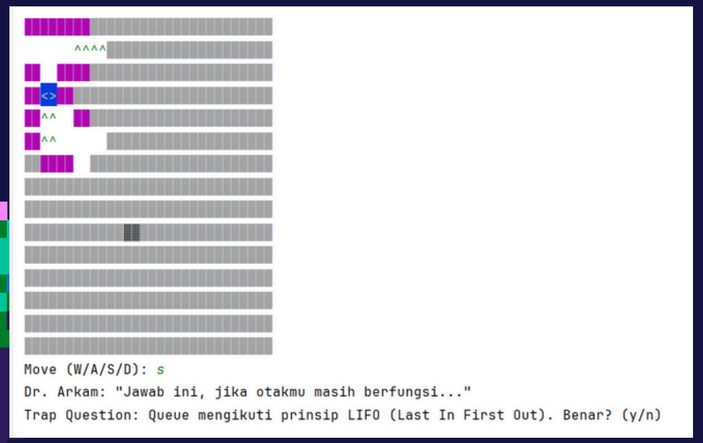
Tembok, Anda tidak bisa berjalan melewati ini

^^ Trap, jika Anda melewati ini, maka Anda diharuskan menjawab pertanyaan



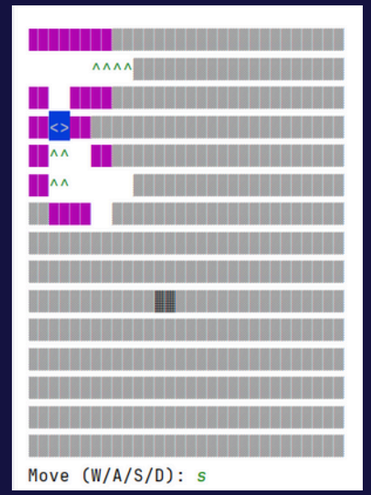


TRAP



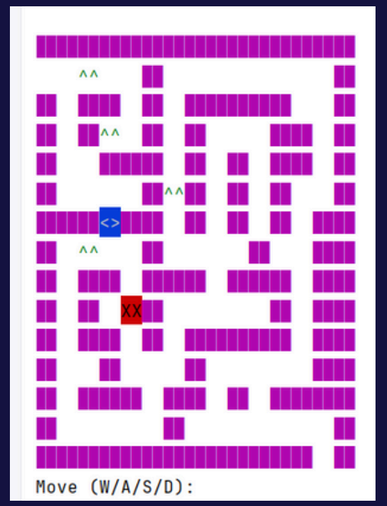
- Terdapat trap berisi pertanyaan materi ASD yang muncul sebagai media pembelajaran yang lebih interaktif dan menantang.
- Jawaban salah akan memberi penalty skip turn
- Trap akan di beberapa kali turn

FOG



- Pada saat tertentu, player tidak bisa melihat keseluruhan labirin (hanya area dekat dirinya yang terlihat)
- Kabut akan hilang di setiap 5 turn dan akan muncul lagi di turn selanjutnya.
- area berwarna lebih gelap merupakan posisi monster

MONSTER



- Monster mengejar1 langkah per turn melalui jalan tercepat menuju player

ALGORITMA

(BFS) - MONSTER PATHFINDING

BFS digunakan agar monster selalu bergerak melalui jalur terpendek menuju pemain.

Cara kerjanya:

1. Dimulai dari posisi monster.
2. Menjelajah grid secara level-by-level.
3. BFS menyimpan parent untuk setiap tile, sehingga bisa ditelusuri kembali jalurnya.
4. Ketika BFS mencapai posisi pemain, BFS berhenti.
5. Monster menelusuri kembali jalur terpendek dari player ke parent hingga menemukan langkah berikutnya.

Kenapa BFS?
Karena BFS menjamin menemukan jalur terpendek pada grid tanpa bobot.

(DFS) - FOG

DFS digunakan untuk membuka area penglihatan di sekitar pemain.

Pada turn ke-1 dan setiap turn ke-5, seluruh peta akan terbuka sementara sebagai efek “vision burst”.

Cara kerjanya:

- Fungsi DFS dipanggil dari posisi player.
- DFS menyebar ke empat arah (atas, bawah, kiri, kanan).
- Kedalaman dibatasi hingga radius tertentu (depth ≤ 5).
- Jika menemukan tembok, DFS berhenti di arah itu.
- Tile yang dilewati DFS menjadi terlihat (visible).

Fungsi DFS cocok untuk ini karena traversal menyelam ke satu jalur hingga batas kedalaman, sesuai konsep radius penglihatan.

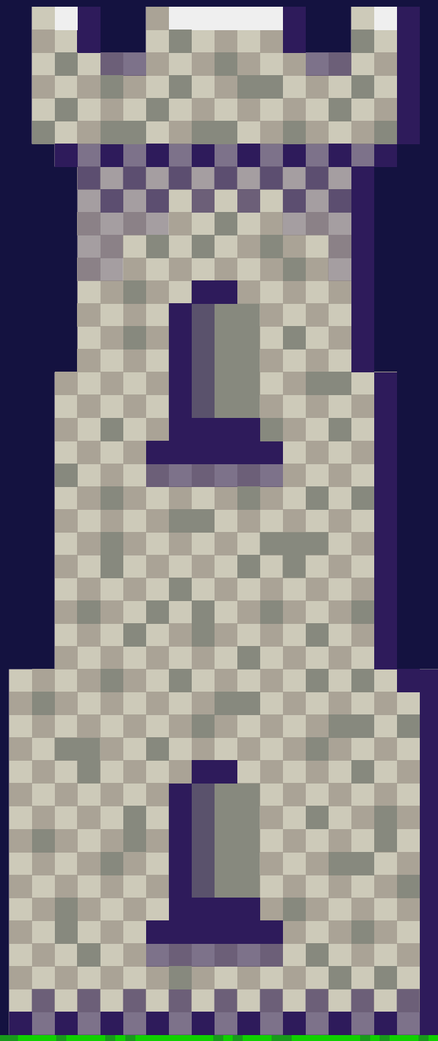
DATA STRUCTURES (COMPLICATED)

01 THE MAP

02 THE MONSTER'S BRAIN (BFS)

03 THE TRAP SYSTEMS

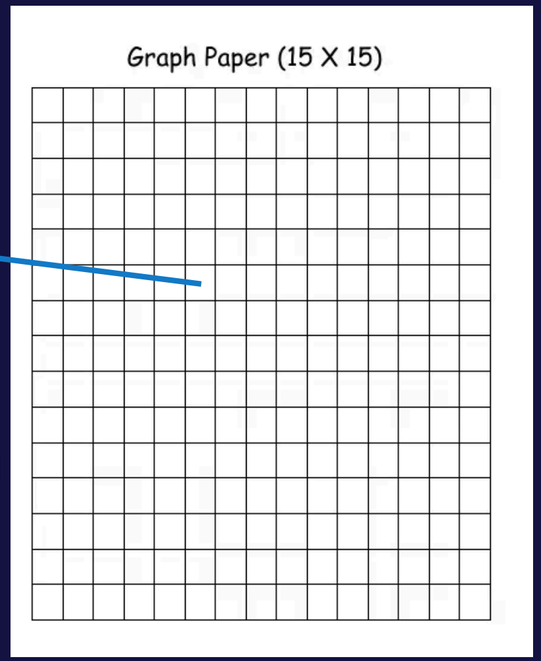
04 THE FOG (DFS)



DATA STRUCTURES USED:

- Array 2D (Tile[][] grid)
- Class Objects (Position, Tile, Entity)
- Queue<Position> → BFS Frontier
- LinkedList
- Boolean[][] visited
- Position[][] parent
- List<Position> (ArrayList)

THE MAP



- Structure: 15x15 Character Matrix (char[][]).
- Memory Efficiency: Stores static map data (Walls vs. Paths).
- Access Speed: O(1) Time Complexity (Instant Lookup).
- Boundary Validation: Prevents IndexOutOfBounds errors.

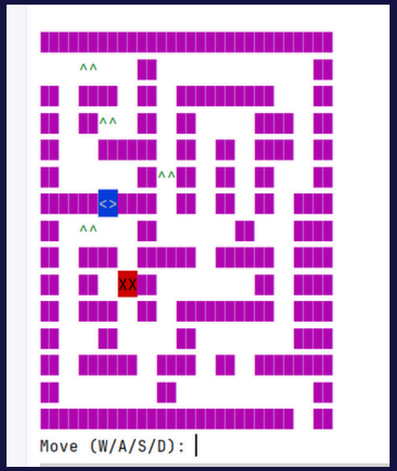
- Dunia permainan kami, 'MonsterChase,' dibangun berdasarkan struktur **data Array 2-Dimensional.**

- Kami memilih Array 2D daripada graph atau LinkedList karena Array dapat melakukan performa melalui **Akses Langsung**.

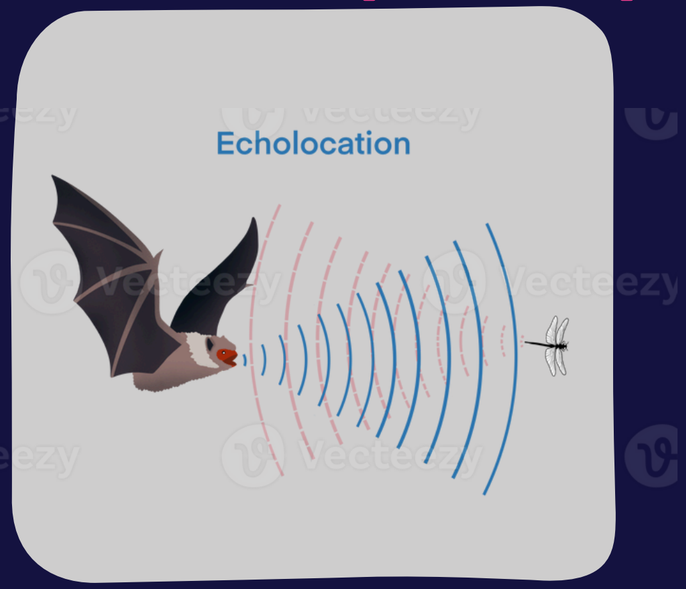
- Dalam permainan kami, Monster perlu terus-menerus memeriksa sekitarnya, dan Player bergerak dengan sering. Dengan Array 2D, kita dapat mengakses tile tertentu, seperti baris 5, kolom 5, secara instan menggunakan koordinatnya. Ini adalah **operasi O(1) atau 'Constant Time'.**

- Jika kami menggunakan Linked List, kami harus menelusuri seluruh daftar hanya untuk mengetahui apakah ada dinding di lokasi tertentu. Dengan matriks array ini, kita cukup memeriksa grid[x][y]. Jika nilainya adalah simbol 'tembok' (|), kita tahu itu adalah dinding dan menghalangi pergerakan.

- Struktur ini juga berfungsi sebagai sistem koordinat kami, memastikan bahwa entitas seperti Player dan Monster selalu dipantau dalam batas valid dari 0 sampai 14.



THE MONSTER'S BRAIN (BFS)



VISUAL

- Algorithm: Breadth-First Search
- Data Structure: Queue
- Objective: Finds the Guaranteed Shortest Path to the target
- Logic: Explores neighbor-by-neighbor (Layer-based search) rather than deep diving into one path

- Untuk implementasi kecerdasan buatan (AI) monster, kami menggunakan struktur data Queue. Tidak seperti Stack yang menelusuri satu jalur secara mendalam, Queue mengikuti kebijakan 'Masuk Pertama, Keluar Pertama' (First-In, First-Out). Ini memungkinkan monster untuk memindai peta lapis demi lapis.

- Berikut cara kerjanya:
Monster memulai dari posisi saat ini (Jarak 0).
Ia memasukkan semua tetangga terdekat ke dalam Queue dan menandainya sebagai 'Jarak 1'.
Kemudian ia memproses tetangga tersebut untuk menemukan 'Jarak 2', dan seterusnya.

- Ini menciptakan efek 'flood fill' yang menyebar di seluruh peta. Saat 'gelombang' ini mencapai koordinat Pemain, algoritma berhenti. Karena kami memperluasnya lapis demi lapis, kami secara matematis dijamin bahwa jalur yang ditemukan adalah Jalur Terpendek yang Mungkin.

- Jika kami menggunakan DFS (Depth-First Search) di sini, monster mungkin akan mengambil rute yang panjang dengan step-step tambahan. BFS memastikan bahwa ia adalah pemburu yang efisien.

THE TRAP SYSTEM

- ArrayList (activeTraps):

Purpose: Maintains the list of currently active traps on the map.
Operation: Linear Traversal O(N). We must check every trap to decrease its timer.

- HashMap (trapQuestions):

Purpose: Links a specific coordinate to a specific quiz question.
Operation: Key-Value Lookup O(1). When a player steps on (x,y), we need the question immediately without searching a list.

THE FOG (DFS)

- Algorithm: Depth-First Search (DFS).
- Data Structure: Implicit Stack (System Call Stack).
- Mechanism: Recursion.
- Logic: The light travels deep into one direction until it hits a wall or runs out of energy (Depth Limit = 5).

- Unlike the Monster, which spreads out in a circle indefinitely (BFS), cahaya dalam game kita berperilaku seperti sinar yang bergerak hingga mengenai rintangan. Untuk mencapai hal ini, kita menggunakan Rekursi.

- Sekarang, Anda tidak akan melihat objek 'Stack' dalam kode kita, tetapi itu tidak berarti kita tidak menggunakannya. Dengan menggunakan fungsi rekursif, di mana fungsi memanggil dirinya sendiri, kita memanfaatkan System Call Stack (atau Implicit Stack) bawaan komputer.

- Cahaya dimulai dari pemain.
- Cahaya tersebut mendorong koordinat berikutnya ke Call Stack dan menyelam lebih dalam.
- Cahaya terus bergerak hingga mengenai dinding atau mencapai kedalaman maksimum 5 tiles.
- Setelah mencapai batas tersebut (Kasus Dasar), cahaya 'keluar' dari stack dan kembali untuk menjelajahi arah lain. Ini memungkinkan kita untuk membuat bentuk pencahayaan yang kompleks tanpa menulis logika perulangan yang kompleks.


THANK
YOU