Proyek I Kecerdasan Buatan

The Diamond Heist: AI vs. Human (Implementasi Algoritma Pencarian Jalur: Breadth-First Search (BFS) vs. A* Search)

Penyusun:

Elkana Sitorus - 11S23009

Wesly Ambarita - 11S23013

Aron Hutapea - 11S23022

Firman Hutasoit - 11S23041

INSTITUT TEKNOLOGI DEL

FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO

I. Deskripsi Singkat Masalah yang Diselesaikan

Proyek ini mengimplementasikan sebuah permainan sederhana di mana pemain manusia (penyusup) harus mengambil berlian (**Diamond**) dan melarikan diri melalui pintu keluar (**Exit**) di dalam sebuah labirin acak. Masalah utama yang diselesaikan adalah **pencarian jalur (pathfinding)** di lingkungan dinamis.

Dua agen AI, yang bertindak sebagai penjaga (**Guard**), ditugaskan untuk mengejar pemain. Masing-masing penjaga menggunakan algoritma pencarian jalur yang berbeda untuk menemukan rute optimal atau tercepat menuju posisi pemain:

1. **Penjaga Merah (Red Guard):** Menggunakan algoritma **Breadth-First Search (BFS)**.
2. **Penjaga Biru (Blue Guard):** Menggunakan algoritma **A* Search**.

Permainan ini berfungsi sebagai platform untuk **membandingkan performa** kedua algoritma pencarian (BFS dan A*) dalam konteks waktu nyata (real-time) di sebuah labirin berbiaya (berkarpet) dan tidak berbiaya (lantai biasa).

II. Penjelasan Implementasi

II.I. Struktur Data yang Digunakan

Struktur Data	**Deskripsi dan Implementasi dalam Kode**
Grid 2D (List of Lists)	Struktur data utama yang direpresentasikan oleh variabel grid. Ini adalah daftar dari daftar objek **Node**, yang secara efektif membentuk labirin (peta permainan). grid[row][col] mengakses sel atau node tertentu.

Kelas Node	Setiap sel dalam grid adalah objek Node. Menyimpan informasi penting seperti: - row, col: Posisi di grid. - is_wall_flag: Menandakan apakah sel adalah tembok (True) atau lantai (False). - traversal_cost: Biaya untuk melintasi node (1 untuk lantai biasa, 5 untuk karpet/rumput). - neighbors: Daftar node yang dapat diakses dari node saat ini. - Variabel Pathfinding: g (biaya dari awal), f (biaya total A*), parent (untuk rekonstruksi jalur).
Queue (queue.Queue)	Digunakan dalam algoritma **BFS** untuk menyimpan node yang akan dieksplorasi. Properti **First-In, First-Out (FIFO)** memastikan eksplorasi berdasarkan kedalaman yang sama (lapisan demi lapisan).
Priority Queue (queue.PriorityQueue)	Digunakan dalam algoritma **A* Search** untuk menyimpan node yang akan dieksplorasi. Menyimpan tuple (f_score, count, node) dan secara otomatis mengambil node dengan nilai **f** (biaya total yang diperkirakan) terkecil, memprioritaskan jalur yang paling menjanjikan.
Set (open_set_hash dan visited)	Digunakan dalam A* dan BFS untuk melacak node yang telah dimasukkan ke dalam Priority Queue/Queue atau yang telah dikunjungi, untuk mencegah pemrosesan berulang.

II.II. Logika Algoritma yang Diimplementasikan

A. Breadth-First Search (BFS)

Fungsi: bfs(grid, start, end)

- **Tujuan:** Menemukan jalur terpendek (dalam jumlah langkah/simpul) antara start dan end tanpa mempertimbangkan biaya lintasan.
- **Logika:**
    1. Memulai dengan queue yang berisi start node.
    2. Menggunakan visited set untuk menghindari simpul yang dikunjungi berulang kali.
    3. Iteratif mengeluarkan node dari queue dan memeriksa semua **tetangganya**.
    4. Jika tetangga belum dikunjungi dan bukan tembok, ia ditambahkan ke visited set, diatur **parent-nya** ke node saat ini, dan dimasukkan ke dalam queue.
    5. Pencarian berhenti ketika end node ditemukan.
- **Peran Penjaga (Guard):** Penjaga BFS akan selalu mengambil rute dengan jumlah langkah paling sedikit, mengabaikan biaya karpet.

B. A* Search

Fungsi: a_star(grid, start, end)

- **Tujuan:** Menemukan jalur dengan biaya total terendah dari start ke end.
- **Logika:** A* adalah ekstensi dari Dijkstra's Algorithm yang menggunakan **fungsi heuristik** untuk memandu pencarian.
    - **Fungsi Biaya (f):** $f(n) = g(n) + h(n)$
        - $g(n)$: **Biaya aktual** untuk bergerak dari node awal ke node n (akumulasi traversal_cost).
        - $h(n)$: **Heuristik**, estimasi biaya untuk bergerak dari node n ke node tujuan. Dalam kode ini, digunakan **Manhattan Distance** (h function).
    - **Heuristik (h):** abs(x1 - x2) + abs(y1 - y2). Ini adalah jarak non-diagonal dari node saat ini ke tujuan.
    - Menggunakan **Priority Queue** untuk selalu menjelajahi node dengan nilai f terendah, memprioritaskan node yang dekat dengan tujuan dan memiliki biaya perjalanan yang rendah.
- **Peran Penjaga (Guard):** Penjaga A* adalah agen AI yang "pintar". Ia akan mencari rute terpendek sambil **mempertimbangkan biaya lintasan**. Karena traversal_cost karpet adalah 5, Penjaga A* akan cenderung menghindari area karpet kecuali jika itu adalah satu-satunya cara untuk mencapai pemain.

III. Analisis dan Perbandingan

Proyek ini membandingkan BFS dan A* secara implisit melalui skenario permainan dengan berbagai tingkat kesulitan:

Tingkat Kesulitan	**Konfigurasi Penjaga**	**Frekuensi Gerak Penjaga**
EASY	1 Penjaga BFS (Merah)	Bergerak 1x setiap 2 langkah pemain
MEDIUM	1 Penjaga BFS (Merah) & 1 Penjaga A* (Biru)	Bergerak 1x setiap 1 langkah pemain
HARD	1 Penjaga BFS (Merah) & 1 Penjaga A* (Biru)	**A* bergerak 2x** setiap 1 langkah pemain

III.I. Perbandingan Performa Algoritma

A. Panjang Jalur (Path Length) & Biaya

- **BFS (Penjaga Merah):** Hanya mempertimbangkan **jumlah simpul** yang dieksplorasi/langkah, mengabaikan traversal_cost. Hasilnya adalah jalur terpendek dalam hal langkah.
- **A* (Penjaga Biru):** Mempertimbangkan **biaya lintasan (traversal_cost)**. Dalam peta yang memiliki karpet (cost=5), Penjaga A* akan cenderung memilih jalur yang lebih panjang dalam langkah (misalnya, lantai biasa cost=1) jika total biayanya lebih rendah daripada jalur yang lebih pendek tetapi melewati karpet.

Analisis: Penjaga A* akan menunjukkan **akurasi** yang lebih baik dalam mencari jalur "termurah" (mengurangi waktu yang dihabiskan di karpet), sementara Penjaga BFS hanya mencari jalur "tercepat" dalam hitungan langkah.

B. Jumlah Simpul yang Dieksplorasi (Waktu Eksekusi)

- **BFS:** Melakukan eksplorasi **buta (uninformed search)**; ia harus memeriksa setiap simpul di setiap "lapisan" dari node awal hingga node tujuan ditemukan. Ini berarti dalam labirin besar, ia akan mengeksplorasi **lebih banyak simpul** secara keseluruhan sebelum mencapai tujuan.
- **A**:** Melakukan eksplorasi **terinformasi (informed search)** berkat fungsi heuristik. Fungsi $h(n)$ memandu pencarian langsung ke arah tujuan. Hasilnya, A* umumnya akan mengeksplorasi **jauh lebih sedikit simpul** daripada BFS.

Analisis: Walaupun A* memiliki komputasi yang sedikit lebih kompleks di setiap simpul (menghitung $f=g+h$ dan mengelola Priority Queue), ia **secara signifikan lebih cepat** dalam menemukan jalur di labirin besar karena pengurangan simpul yang dieksplorasi. Ini terlihat jelas pada tingkat kesulitan **HARD**, di mana Penjaga A* diizinkan bergerak dua kali dalam satu giliran pemain, menegaskan superioritas kecepatannya.

C. Tingkat Keberhasilan (Konteks Permainan)

- **Tingkat EASY (BFS only):** Pemain memiliki tingkat keberhasilan tinggi karena Penjaga BFS lambat (bergerak 1x setiap 2 langkah) dan cenderung "bodoh" (mengabaikan karpet/lantai berbiaya tinggi) jika jalur terpendek adalah melalui karpet.
- **Tingkat MEDIUM/HARD (BFS & A*):** Tingkat keberhasilan pemain menurun drastis. Penjaga A* adalah ancaman utama karena:
    1. **Kepintaran:** Ia memilih rute biaya terendah.
    2. **Kecepatan:** Ia secara efektif memproses jalur lebih cepat (waktu eksekusi internal), dan pada HARD, ia mendapat bonus 2x langkah.

Analisis Mengapa Hasil Demikian: Perbedaan kinerja disebabkan oleh penggunaan **heuristik** pada A*. Heuristik (Manhattan Distance) memberikan perkiraan seberapa jauh tujuan, memungkinkan A* untuk memotong sebagian besar ruang pencarian. BFS tidak memiliki panduan ini dan harus menjelajahi secara menyeluruh.

IV. Kesimpulan

Proyek "The Diamond Heist" berhasil memvisualisasikan dan membandingkan secara efektif dua algoritma pencarian jalur fundamental: **Breadth-First Search (BFS)** dan **A* Search**.

1. **BFS** adalah algoritma pencarian simpul-terpendek (langkah) yang andal, cocok untuk labirin dengan biaya lintasan seragam, tetapi performanya lambat karena harus menjelajahi seluruh ruang pencarian secara sistematis.
2. **A*** adalah algoritma pencarian biaya-terendah yang jauh lebih efisien. Dengan memanfaatkan **fungsi heuristik**, A* dapat memandu pencarian menuju tujuan, menghasilkan waktu eksekusi yang lebih cepat dan jalur yang **lebih optimal** (biaya terendah) di lingkungan berbiaya (seperti karpet dengan biaya 5).

Dalam konteks permainan, Penjaga A* yang "pintar" (mempertimbangkan biaya) dan "cepat" (memiliki eksplorasi simpul yang lebih sedikit) terbukti menjadi ancaman AI yang jauh lebih superior daripada Penjaga BFS yang hanya mencari langkah terpendek. Proyek ini membuktikan bahwa untuk masalah pencarian jalur dengan biaya lintasan yang berbeda, **A* Search adalah pilihan algoritma yang jauh lebih unggul**.

V. Lampiran: Kode Program

```
# Nama File: diamond_heist_final_v6.7.py

# Proyek 1: "The Diamond Heist: AI vs. Human"

# Versi 6.7: Latar Belakang Menu Utama



import pygame

import queue

import math

import time

import random

import sys

import os



# --- KONFIGURASI TAMPILAN (PYGAME) ---
```

```
WIDTH = 800

ROWS = 25

WIN = pygame.display.set_mode((WIDTH, WIDTH))

pygame.display.set_caption("Proyek 1 AI: The Diamond Heist (BFS vs
A*)")

pygame.font.init()

GAME_FONT = pygame.font.SysFont('Arial', 30, bold=True)

INFO_FONT = pygame.font.SysFont('Arial', 24)



# --- DEFINISI WARNA ---

WARNA_PUTIH = (255, 255, 255)

WARNA_HITAM = (0, 0, 0)

WARNA_BIRU = (0, 100, 255)

WARNA_HIJAU = (0, 200, 0)

WARNA_DIAMOND = (0, 255, 255)

WARNA_ABU_ABU = (128, 128, 128)

WARNA_AI_BFS = (255, 0, 0)

WARNA_AI_ASTAR = (0, 174, 255)

# WARNA_KARPET (Tidak terpakai)




# --- (BARU) LOKASI FOLDER ASET ---

ASSET_DIR = "assets"
```

```python
# --- FUNGSI PEMBUAT SPRITE ---


def load_scaled_png(filename, size):
    """
    (FUNGSI HELPER BARU)
    Memuat file PNG dari folder ASSET_DIR, menskalakan,
    dan mengatur transparansi (colorkey HITAM).
    """
    try:
        filepath = os.path.join(ASSET_DIR, filename)
        # .convert() + set_colorkey() untuk background solid
        image = pygame.image.load(filepath).convert()
        image.set_colorkey(WARNA_HITAM) # Gunakan ini jika BG gambar
Anda HITAM


        return pygame.transform.scale(image, (size, size))
    except pygame.error as e:
        print(f"--- ERROR ---")
        print(f"Tidak dapat memuat gambar: {filepath}")
        print(f"Pastikan folder 'assets' ada dan berisi file
'{filename}'")
        print(f"Error Pygame: {e}")
        print(f"---------------")
        error_surf = pygame.Surface((size, size))
        error_surf.fill((255, 0, 255))
        return error_surf
```

```
    except FileNotFoundError:
        print(f"--- ERROR ---")
        print(f"File tidak ditemukan: {filepath}")
        print(f"Pastikan folder 'assets' ada dan berisi file
'{filename}'")
        print(f"---------------")
        error_surf = pygame.Surface((size, size))
        error_surf.fill((255, 0, 255))
        return error_surf

def create_exit_sprite(size):
    """(TETAP SAMA) Membuat sprite pintu keluar secara prosedural."""
    surf = pygame.Surface((16, 16)); door_color = WARNA_HIJAU;
frame_color = (0, 100, 0); knob_color = (255, 223, 0)
    pygame.draw.rect(surf, frame_color, (3, 1, 10, 15));
pygame.draw.rect(surf, door_color, (4, 2, 8, 13))
    pygame.draw.rect(surf, knob_color, (10, 7, 2, 2))
    surf.set_colorkey(WARNA_HITAM)
    return pygame.transform.scale(surf, (size, size))

# --- ---------------------------- ---

# --- KELAS UTAMA: Node / Sel (TETAP SAMA v6.6) ---
class Node:
    def __init__(self, row, col, width, total_rows, sprites):
```

```
        self.row = row; self.col = col; self.x = row * width; self.y =
col * width

        self.width = width; self.total_rows = total_rows

        self.sprites_dict = sprites

        self.sprite = None

        self.color = None

        self.is_wall_flag = False

        self.neighbors = []; self.traversal_cost = 1; self.g =
float("inf")

        self.f = float("inf"); self.parent = None

        self.make_floor()

    def get_pos(self): return self.row, self.col

    def is_wall(self): return self.is_wall_flag

    def reset_path_vars(self): self.g = float("inf"); self.f =
float("inf"); self.parent = None

    def make_wall(self):

        self.sprite = self.sprites_dict["wall"]

        self.color = None

        self.is_wall_flag = True

        self.traversal_cost = float("inf")

    def make_floor(self):

        self.sprite = None

        self.color = None

        self.is_wall_flag = False

        self.traversal_cost = 1

    def make_carpet(self):
```

```
        self.sprite = self.sprites_dict["grass"]

        self.color = None

        self.is_wall_flag = False

        self.traversal_cost = 5

    def draw(self, win):

        if self.sprite:

            win.blit(self.sprite, (self.x, self.y))

        pass

    def update_neighbors(self, grid):

        self.neighbors = []

        if self.row < self.total_rows - 1 and not grid[self.row +
1][self.col].is_wall(): self.neighbors.append(grid[self.row +
1][self.col])

        if self.row > 0 and not grid[self.row - 1][self.col].is_wall():
self.neighbors.append(grid[self.row - 1][self.col])

        if self.col < self.total_rows - 1 and not
grid[self.row][self.col + 1].is_wall():
self.neighbors.append(grid[self.row][self.col + 1])

        if self.col > 0 and not grid[self.row][self.col - 1].is_wall():
self.neighbors.append(grid[self.row][self.col - 1])

    def __lt__(self, other): return self.f < other.f



# --- KELAS ENTITAS GAME (TETAP SAMA v6.6) ---

class Player:

    def __init__(self, row, col, frames_list):

        self.row = row

        self.col = col
```

```
        self.frames = frames_list
        self.is_moving = False
        self.frame_index = 0
        self.animation_speed = 100
        self.last_update_time = pygame.time.get_ticks()
        self.sprite = self.frames[self.frame_index]
        self.has_diamond = False
    def move(self, dr, dc, grid):
        new_row, new_col = self.row + dr, self.col + dc
        if 0 <= new_row < len(grid) and 0 <= new_col < len(grid[0]):
            if not grid[new_row][new_col].is_wall():
                self.row = new_row
                self.col = new_col
                self.is_moving = True
                return True
        self.is_moving = False
        return False
    def update_animation(self):
        now = pygame.time.get_ticks()
        if now - self.last_update_time > self.animation_speed:
            self.last_update_time = now
            self.frame_index = (self.frame_index + 1) %
len(self.frames)
        self.sprite = self.frames[self.frame_index]
        self.is_moving = False
```

```
    def draw(self, win, width):
        win.blit(self.sprite, (self.row * width, self.col * width))


class Guard:
    def __init__(self, row, col, frames_list, algo_type):
        self.row = row
        self.col = col
        self.frames = frames_list
        self.algo_type = algo_type
        self.frame_index = 0
        self.animation_speed = 150
        self.last_update_time = pygame.time.get_ticks()
        self.sprite = self.frames[self.frame_index]
    def move(self, grid, player_node):
        start_node = grid[self.row][self.col]
        end_node = player_node
        path = None
        if self.algo_type == 'BFS': path = bfs(grid, start_node,
end_node)
        elif self.algo_type == 'A*': path = a_star(grid, start_node,
end_node)
        if path and len(path) > 1:
            next_node = path[1]
            self.row = next_node.row
            self.col = next_node.col
    def update_animation(self):
```

```
        now = pygame.time.get_ticks()
        if now - self.last_update_time > self.animation_speed:
            self.last_update_time = now
            self.frame_index = (self.frame_index + 1) %
len(self.frames)
        self.sprite = self.frames[self.frame_index]
    def draw(self, win, width):
        win.blit(self.sprite, (self.row * width, self.col * width))


class AnimatedItem:
    def __init__(self, row, col, frames_list):
        self.row = row
        self.col = col
        self.frames = frames_list
        self.frame_index = 0
        self.animation_speed = 300
        self.last_update_time = pygame.time.get_ticks()
        self.sprite = self.frames[self.frame_index]
    def update_animation(self):
        now = pygame.time.get_ticks()
        if now - self.last_update_time > self.animation_speed:
            self.last_update_time = now
            self.frame_index = (self.frame_index + 1) %
len(self.frames)
            self.sprite = self.frames[self.frame_index]
    def draw(self, win, width):
```

```
        win.blit(self.sprite, (self.row * width, self.col * width))


# --- FUNGSI PATHFINDING (TETAP SAMA v6.6) ---
def h(p1, p2):
    x1, y1 = p1; x2, y2 = p2; return abs(x1 - x2) + abs(y1 - y2)
def reset_pathfinding_vars(grid):
    for row in grid:
        for node in row: node.reset_path_vars()
def reconstruct_path(current_node):
    path = [];
    while current_node: path.append(current_node); current_node =
current_node.parent
    return path[::-1]


def bfs(grid, start, end):
    reset_pathfinding_vars(grid); q = queue.Queue(); q.put(start)
    visited = {start}; start.g = 0
    while not q.empty():
        current = q.get()
        if current == end: return reconstruct_path(current)
        for neighbor in current.neighbors:
            if neighbor not in visited:
                neighbor.parent = current; neighbor.g = current.g + 1
                visited.add(neighbor); q.put(neighbor)
    return None
```

```
def a_star(grid, start, end):
    reset_pathfinding_vars(grid); count = 0; open_set =
queue.PriorityQueue()
    open_set.put((0, count, start)); open_set_hash = {start}
    start.g = 0; start.f = h(start.get_pos(), end.get_pos())
    while not open_set.empty():
        current = open_set.get()[2]; open_set_hash.remove(current)
        if current == end: return reconstruct_path(current)
        for neighbor in current.neighbors:
            temp_g_score = current.g + neighbor.traversal_cost
            if temp_g_score < neighbor.g:
                neighbor.parent = current; neighbor.g = temp_g_score
                neighbor.f = temp_g_score + h(neighbor.get_pos(),
end.get_pos())
                if neighbor not in open_set_hash:
                    count += 1; open_set.put((neighbor.f, count,
neighbor)); open_set_hash.add(neighbor)
    return None


# --- FUNGSI SETUP & DRAWING GAME ---


def make_grid(rows, width, sprites):
    grid = []; gap = width // rows
    for i in range(rows):
        grid.append([])
```

```
        for j in range(rows):
            node = Node(i, j, gap, rows, sprites); grid[i].append(node)
    return grid


def draw_grid_lines(win, rows, width):
    gap = width // rows
    for i in range(rows): pygame.draw.line(win, WARNA_ABU_ABU, (0, i * gap), (width, i * gap))
    for j in range(rows): pygame.draw.line(win, WARNA_ABU_ABU, (j * gap, 0), (j * gap, width))


def draw_game(win, grid, rows, width, player, guards, diamond_obj, exit_pos, difficulty, sprites):
    # (TETAP SAMA v6.6)
    win.fill(WARNA_PUTIH)
    draw_grid_lines(win, rows, width)
    for row in grid:
        for node in row:
            node.draw(win)
    gap = width // rows
    win.blit(sprites["exit"], (exit_pos[0] * gap, exit_pos[1] * gap))
    if not player.has_diamond:
        diamond_obj.draw(win, gap)
    player.draw(win, gap)
    for guard in guards: guard.draw(win, gap)
    if difficulty == "EASY":
```

```
        info_text_bfs = INFO_FONT.render("Merah = BFS (Lambat)", True,
WARNA_AI_BFS); win.blit(info_text_bfs, (10, 10))

    else:

        info_text_bfs = INFO_FONT.render("Merah = BFS (Bodoh)", True,
WARNA_AI_BFS); win.blit(info_text_bfs, (10, 10))

        if difficulty == "HARD": info_text_astar =
INFO_FONT.render("Biru = A* (SANGAT CEPAT)", True, WARNA_AI_ASTAR)

        else: info_text_astar = INFO_FONT.render("Biru = A* (Pintar)",
True, WARNA_AI_ASTAR)

        win.blit(info_text_astar, (10, 35))

    pygame.display.update()



# --- PETA ACAK (TETAP SAMA v6.6) ---

def create_random_map(grid, safe_zones, difficulty):

    if difficulty == "EASY":

        NUM_WALLS = 160

        NUM_CARPET_PATCHES = 5

        MAX_CARPET_SIZE = 20

        print_msg = "Peta acak 'Mudah' telah dibuat."

    elif difficulty == "HARD":

        NUM_WALLS = 240

        NUM_CARPET_PATCHES = 10

        MAX_CARPET_SIZE = 30

        print_msg = "Peta acak 'Sulit' telah dibuat."

    else: # Medium (Normal)

        NUM_WALLS = 200

        NUM_CARPET_PATCHES = 8
```

```
        MAX_CARPET_SIZE = 25
        print_msg = "Peta acak 'Normal' telah dibuat."
    for i in range(ROWS):
        for j in range(ROWS):
            if i == 0 or i == ROWS - 1 or j == 0 or j == ROWS - 1:
                grid[i][j].make_wall()
    for _ in range(NUM_WALLS):
        row = random.randint(1, ROWS - 2)
        col = random.randint(1, ROWS - 2)
        pos = (row, col)
        if pos not in safe_zones and not grid[row][col].is_wall():
            grid[row][col].make_wall()
    for _ in range(NUM_CARPET_PATCHES):
        row = random.randint(1, ROWS - 2); col = random.randint(1, ROWS
- 2)
        current_size = 0
        while current_size < MAX_CARPET_SIZE:
            pos = (row, col)
            if pos in safe_zones: break
            if (not grid[row][col].is_wall()) and (pos not in
safe_zones):
                grid[row][col].make_carpet(); current_size += 1
            move = random.choice([(0, 1), (0, -1), (1, 0), (-1, 0)]);
row += move[0]; col += move[1]
            if not (1 <= row < ROWS - 2 and 1 <= col < ROWS - 2): break
    for pos in safe_zones:
```

```
        grid[pos[0]][pos[1]].make_floor()
    print(print_msg)

def is_map_connected(grid, all_spawn_points):
    if not all_spawn_points: return True
    start_node = grid[all_spawn_points[0][0]][all_spawn_points[0][1]]
    goal_nodes = {grid[pos[0]][pos[1]] for pos in all_spawn_points[1:]}
    for row in grid:
        for node in row:
            node.update_neighbors(grid)
    q = queue.Queue(); q.put(start_node); visited = {start_node}
    while not q.empty():
        current = q.get()
        if current in goal_nodes: goal_nodes.remove(current)
        for neighbor in current.neighbors:
            if neighbor not in visited and not neighbor.is_wall():
                visited.add(neighbor); q.put(neighbor)
    return len(goal_nodes) == 0

# --- FUNGSI TAMPILAN PESAN & MENU (DIUBAH) ---
def show_message(win, text, title_font, regular_font, bg_sprite=None):
    """
    (DIUBAH)
    Sekarang menerima 'bg_sprite' opsional.
    Jika ada, gambar itu. Jika tidak, isi dengan warna hitam.
```

```
    """
    if bg_sprite:
        win.blit(bg_sprite, (0, 0)) # Gambar latar belakang
    else:
        win.fill(WARNA_HITAM) # Fallback ke layar hitam (untuk Game
Over)

    lines = text.split('\n')
    start_y = (WIDTH // 2) - (len(lines) * 30)
    for i, line in enumerate(lines):
        if line.startswith("[") or line.islower() or
line.startswith("("): font_to_use = regular_font
        else: font_to_use = title_font

        # (DIUBAH) Teks sekarang digambar dengan shadow hitam agar
terbaca
        # di atas background apa pun
        text_surface_shadow = font_to_use.render(line, True,
WARNA_HITAM)
        text_rect_shadow = text_surface_shadow.get_rect(center=(WIDTH
// 2 + 2, start_y + i * 45 + 2))
        win.blit(text_surface_shadow, text_rect_shadow)

        text_surface = font_to_use.render(line, True, WARNA_PUTIH)
        text_rect = text_surface.get_rect(center=(WIDTH // 2, start_y +
i * 45))
        win.blit(text_surface, text_rect)
```

```
    pygame.display.update()


def show_menu_screen(win, bg_sprite):
    """
    (DIUBAH)
    Sekarang menerima 'bg_sprite' dan meneruskannya ke 'show_message'.
    """
    menu_text = (
        "THE DIAMOND HEIST\n\n"
        "PILIH TINGKAT KESULITAN\n\n"
        "[E] Mudah\n"
        "(1 Penjaga BFS, bergerak 1x tiap 2 langkahmu)\n\n"
        "[M] Normal\n"
        "(2 Penjaga, bergerak 1x tiap 1 langkahmu)\n\n"
        "[H] Sulit\n"
        "(2 Penjaga, A* bergerak 2x tiap 1 langkahmu)"
    )
    title_font = pygame.font.SysFont('Arial', 32, bold=True)
    option_font = pygame.font.SysFont('Arial', 24)
    while True:
        # (DIUBAH) Meneruskan bg_sprite ke show_message
        show_message(win, menu_text, title_font, option_font,
bg_sprite=bg_sprite)
```

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT: return None
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_e: return "EASY"
                if event.key == pygame.K_m: return "MEDIUM"
                if event.key == pygame.K_h: return "HARD"
                if event.key == pygame.K_q: return None

# --- FUNGSI SESI GAME ---
def run_game_session(win, width, difficulty, sprites):

    grid = make_grid(ROWS, width, sprites)
    player_move_count = 0

    map_valid = False
    while not map_valid:
        mid_row = ROWS // 2; mid_col = ROWS // 2
        q1 = [(r, c) for r in range(2, mid_row-1) for c in range(2,
mid_col-1)]
        q2 = [(r, c) for r in range(2, mid_row-1) for c in
range(mid_col+1, ROWS-2)]
        q3 = [(r, c) for r in range(mid_row+1, ROWS-2) for c in
range(2, mid_col-1)]
        q4 = [(r, c) for r in range(mid_row+1, ROWS-2) for c in
range(mid_col+1, ROWS-2)]
        try:
```

```
            START_POS = random.choice(q1); DIAMOND_POS =
random.choice(q4)
            GUARD_BFS_POS = random.choice(q2); GUARD_ASTAR_POS =
random.choice(q3)
            EXIT_POS = START_POS
        except IndexError:
            print("Error: Grid terlalu kecil."); return "QUIT"

        all_spawn_points = [START_POS, DIAMOND_POS, GUARD_BFS_POS,
GUARD_ASTAR_POS]
        safe_zones = set(all_spawn_points)

        create_random_map(grid, safe_zones, difficulty)
        map_valid = is_map_connected(grid, all_spawn_points)

        if not map_valid:
            print("Peta tidak valid, membuat ulang...")

    player = Player(START_POS[0], START_POS[1],
sprites["player_frames"])
    guard_bfs = Guard(GUARD_BFS_POS[0], GUARD_BFS_POS[1],
sprites["guard_bfs_frames"], "BFS")
    guard_astar = Guard(GUARD_ASTAR_POS[0], GUARD_ASTAR_POS[1],
sprites["guard_astar_frames"], "A*")
    diamond_obj = AnimatedItem(DIAMOND_POS[0], DIAMOND_POS[1],
sprites["diamond_frames"])

    active_guards = []
```

```
    if difficulty == "EASY": active_guards.append(guard_bfs)
    else: active_guards.append(guard_bfs);
active_guards.append(guard_astar)

    run_game = True; game_over = False; win_message = ""; clock =
pygame.time.Clock()

    print(f"\n--- PERMAINAN BARU DIMULAI (Tingkat: {difficulty}) ---")
    print("Kontrol: Panah (Atas, Bawah, Kiri, Kanan)")

    while run_game:
        clock.tick(30)

        player_moved = False
        for event in pygame.event.get():
            if event.type == pygame.QUIT: return "QUIT"
            if event.type == pygame.KEYDOWN:
                dr, dc = 0, 0
                if event.key == pygame.K_LEFT: dr, dc = -1, 0
                elif event.key == pygame.K_RIGHT: dr, dc = 1, 0
                elif event.key == pygame.K_UP: dr, dc = 0, -1
                elif event.key == pygame.K_DOWN: dr, dc = 0, 1

                if (dr, dc) != (0, 0) and not game_over:
                    player_moved = player.move(dr, dc, grid)
```

```
        player.update_animation()
        diamond_obj.update_animation()

        if player_moved and not game_over:
            player_move_count += 1
            run_ai_turn = False
            if difficulty == "EASY":
                if player_move_count % 2 == 0: run_ai_turn = True
            else: run_ai_turn = True

            if run_ai_turn:
                player_node = grid[player.row][player.col]
                for guard in active_guards:
                    guard.move(grid, player_node)
                    if difficulty == "HARD" and guard.algo_type ==
"A*":
                        guard.move(grid, player_node)

        for guard in active_guards:
            guard.update_animation()

        if not game_over:
            for guard in active_guards:
                if guard.row == player.row and guard.col == player.col:
                    win_message = "ANDA KALAH!"; game_over = True;
run_game = False; break
```

```
            if game_over: continue



            if not player.has_diamond:

                if player.row == DIAMOND_POS[0] and player.col ==
DIAMOND_POS[1]:

                    player.has_diamond = True; print("--- ANDA
MENDAPATKAN BERLIAN! ---")



            if player.has_diamond:

                if player.row == EXIT_POS[0] and player.col ==
EXIT_POS[1]:

                    win_message = "ANDA MENANG!"; game_over = True;
run_game = False



        draw_game(win, grid, ROWS, width, player, active_guards,
diamond_obj, EXIT_POS, difficulty, sprites)



    if game_over:

        msg = f"{win_message}\n\nTekan 'R' untuk Mulai Ulang (Level
Ini)\n\nTekan 'M' untuk kembali ke Menu\n\nTekan 'Q' untuk Keluar"

        # (DIUBAH) Panggilan ini tidak mengirim bg_sprite, jadi akan

        # otomatis menggunakan latar belakang hitam (default).

        show_message(win, msg, GAME_FONT, INFO_FONT)

        while True:

            for event in pygame.event.get():

                if event.type == pygame.QUIT: return "QUIT"

                if event.type == pygame.KEYDOWN:
```

```
                    if event.key == pygame.K_r: return "RESTART"
                    if event.key == pygame.K_m: return "MENU"
                    if event.key == pygame.K_q: return "QUIT"
    return "QUIT"

# --- FUNGSI UTAMA (MAIN LOOP) (DIUBAH) ---
def main(win, width):
    run_app = True

    gap = WIDTH // ROWS

    # --- (BARU) Memuat Latar Belakang Menu ---
    print("Memuat latar belakang menu...")
    bg_sprite = None # Default ke None (layar hitam)
    try:
        bg_path = os.path.join(ASSET_DIR, "bg.png")
        # .convert() standar, karena tidak perlu transparansi
        bg_sprite = pygame.image.load(bg_path).convert()
        bg_sprite = pygame.transform.scale(bg_sprite, (width, width))
        print("Latar belakang menu 'bg.png' dimuat.")
    except Exception as e:
        print(f"--- ERROR Memuat bg.png ---: {e}")
        print("Pastikan 'bg.png' ada di folder 'assets'.")
        print("Fallback: Menggunakan layar hitam untuk menu.")
        # bg_sprite akan tetap None
```

```
    print("Memuat aset gambar...")
    sprites = {
        "player_frames": [
            load_scaled_png("player_0.png", gap),
            load_scaled_png("player_1.png", gap)
        ],
        "guard_bfs_frames": [
            load_scaled_png("guard_bfs_0.png", gap),
            load_scaled_png("guard_bfs_1.png", gap)
        ],
        "guard_astar_frames": [
            load_scaled_png("guard_astar_0.png", gap),
            load_scaled_png("guard_astar_1.png", gap)
        ],
        "diamond_frames": [
            load_scaled_png("diamond_0.png", gap),
            load_scaled_png("diamond_1.png", gap)
        ],
        "exit": create_exit_sprite(gap),
        "grass": load_scaled_png("grass.png", gap), # Ini adalah KARPET
(cost 5)
        "wall": load_scaled_png("wall.png", gap)    # Ini adalah TEMBOK
    }
```

```
    print("Aset berhasil dimuat.")


    while run_app:
        # (DIUBAH) Mengirim bg_sprite ke menu
        difficulty = show_menu_screen(win, bg_sprite)

        if difficulty is None: run_app = False; break

        while True:
            game_result = run_game_session(win, width, difficulty,
sprites)
            if game_result == "QUIT": run_app = False; break
            elif game_result == "MENU": break
            elif game_result == "RESTART": continue

    pygame.quit()
    sys.exit()

# --- ENTRY POINT ---
if __name__ == "__main__":
    main(WIN, WIDTH)
```