

Pencarian Jalur Terpendek Pada Permainan Pacman Menggunakan Algoritma A*



Anggota Kelompok



Salma Aida Y
434221037



Markus Pardede
502320010024



Table of Content

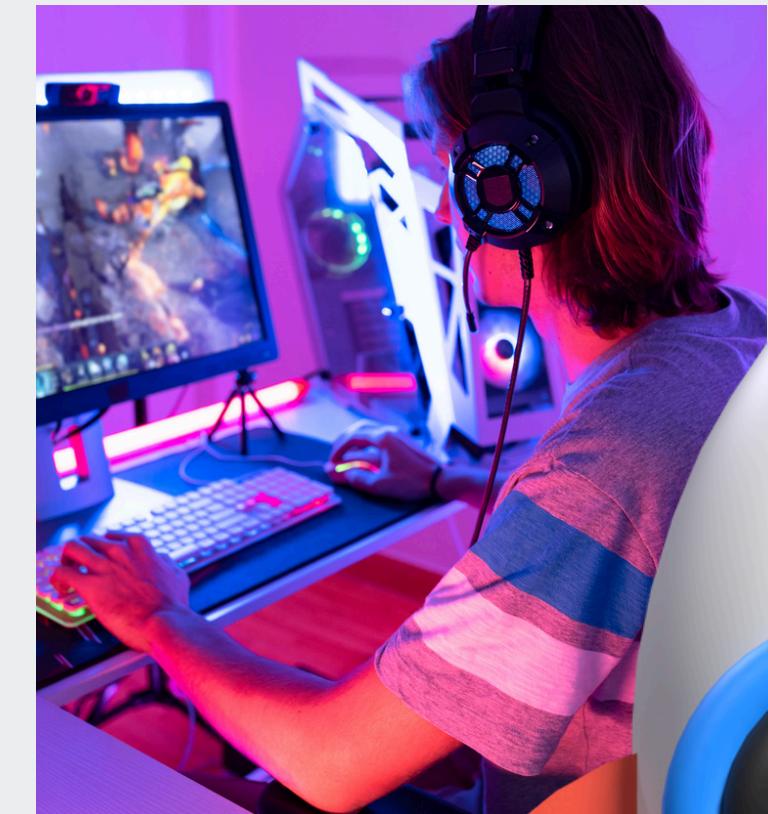
- 01 Pendahuluan
- 02 AI dan Game Search
- 03 Algoritma A*
- 04 Game Pacman
- 05 Implementasi Algoritma
- 06 Dokumentasi dan Daftar Pustaka



Pendahuluan



Memainkan sebuah permainan merupakan salah satu cara yang umumnya kita lakukan untuk melepas penat. Hampir semua permainan saat ini dibuat secara online sehingga dapat diakses oleh siapapun dan kapanpun. Ketika ingin memenangkan sebuah permainan kita dituntut untuk mencari jalur terpendek atau menggunakan algoritma tertentu untuk menentukan bahwa jalur yang dilewati merupakan jalur terbaik dan terpendek. Dengan algoritma juga, para developer game juga dapat memastikan bahwa game dapat berjalan sesuai dengan yang direncanakan dan meminimalisir risiko kegagalan dari game tersebut.





AI dan Game Search

Kecerdasan Buatan (AI) adalah bidang ilmu komputer yang dikhawasukan untuk memecahkan masalah kognitif yang umumnya terkait dengan kecerdasan manusia. Konsep AI pertama kali muncul pada tahun 1956, ketika para ilmuwan merintis langkah pertama dalam pengembangan kecerdasan buatan. Namun, si tahun-tahun berikutnya, perkembangan AI mengalami penurunan minat dan dukungan dan pada tahun 2000-an, kemajuan dalam teknologi komputer dan perkembangan algoritma memulai era baru dalam perkembangan AI.

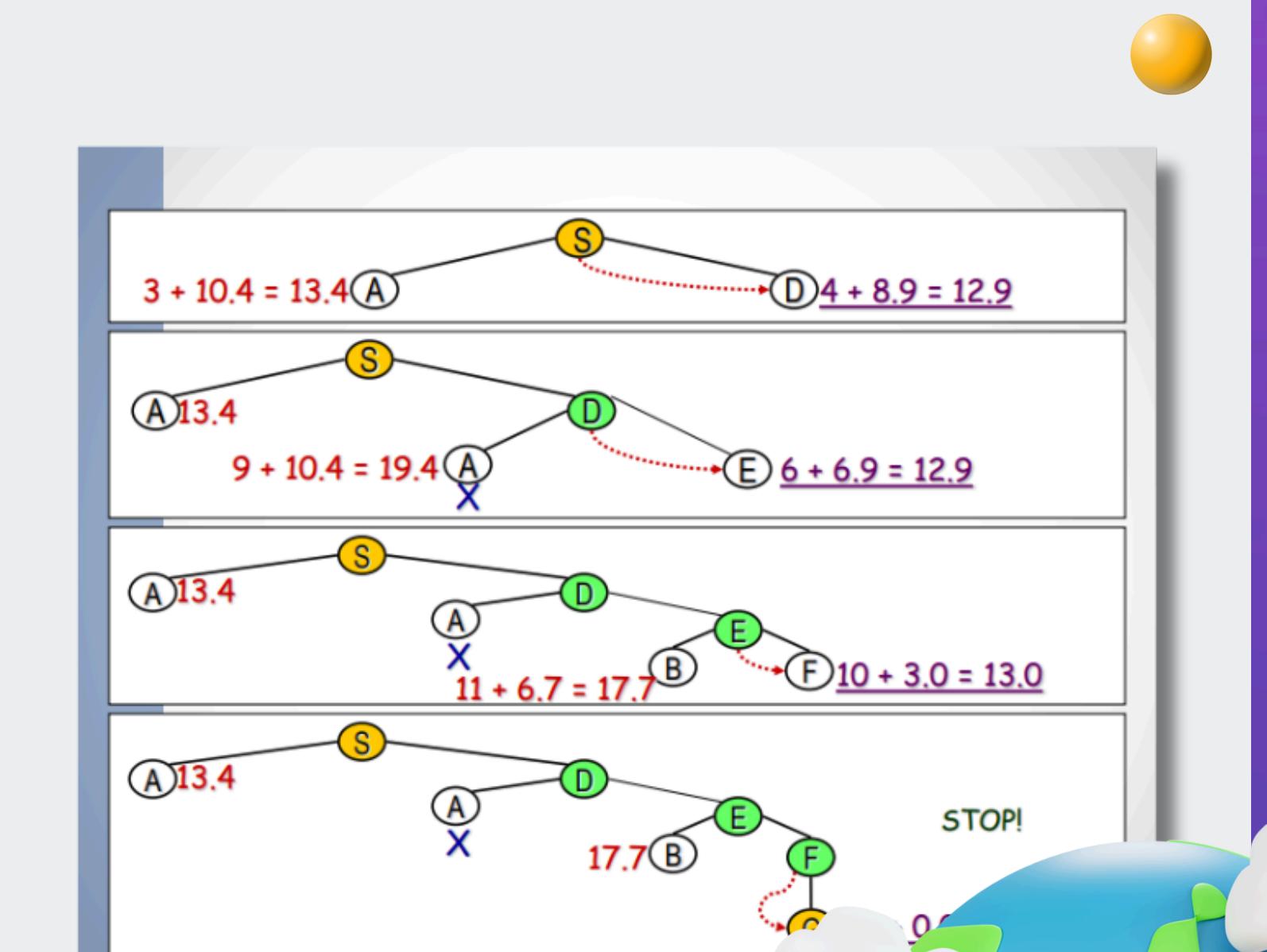


AI dan Game Search

Sedangkan Game Search dalam kecerdasan buatan adalah proses mencari solusi atau langkah terbaik dalam sebuah permainan atau situasi yang terdefinisi dengan baik. Ini melibatkan pencarian melalui ruang keadaan yang mungkin dan pengembangan strategi untuk mencapai tujuan tertentu, seperti memenangkan permainan atau mencapai tujuan tertentu. Algoritma pencarian yang digunakan seperti Minimax, Alpha-Beta Pruning, A*, dan masih banyak lagi

Algoritma A*

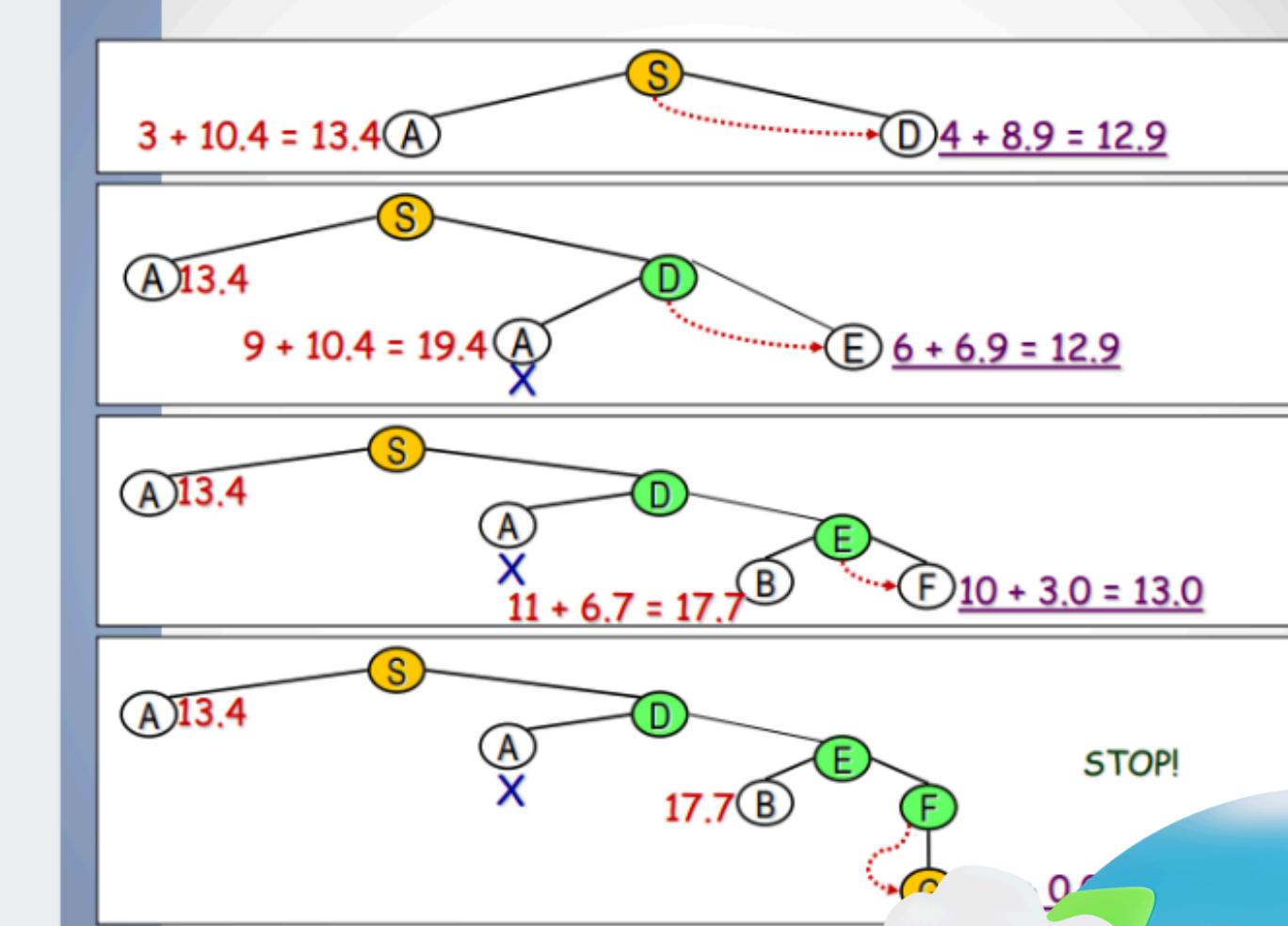
Algoritma A* pertama kali ditemukan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael pada tahun 1968. Algoritma A* adalah algoritma traversal yang digunakan untuk mencari rute terpendek. Ide dari algoritma ini adalah menghindari mengekspansi simpul yang memiliki cost yang besar, algoritma ini juga menerapkan teknik heuristic. Teknik pencarian heuristic sendiri merupakan suatu strategi untuk melakukan proses pencarian ruang keadaan suatu masalah secara efektif.



Algoritma A*

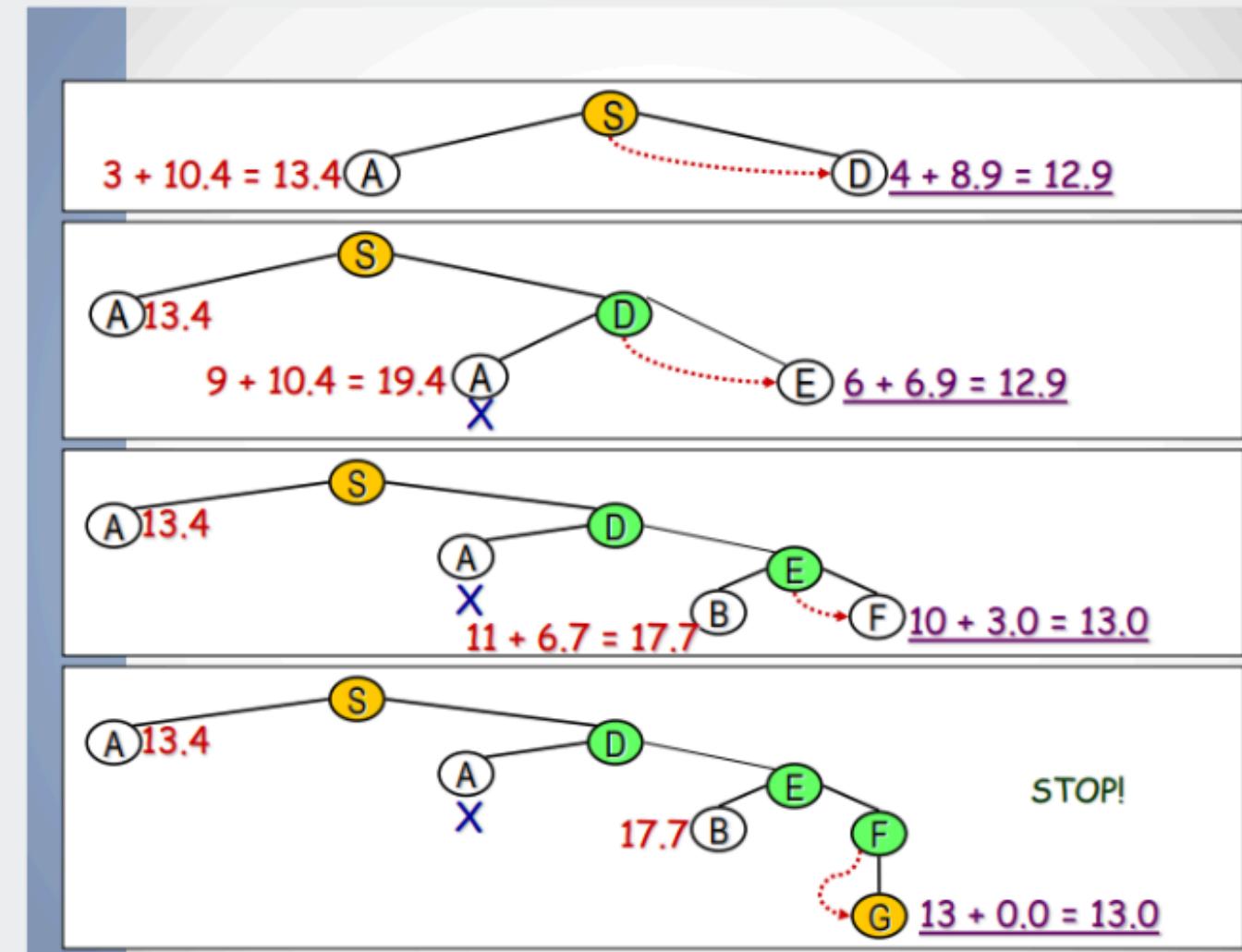
Algoritma ini menyelesaikan permasalahan yang bisa direpresentasikan dengan graf. Setiap langkah akan menjumlahkan $g(n)$ geographical cost, yaitu cost untuk mencapai sebuah node dari akar, dan heuristic cost $h(n)$, yaitu cost untuk mencapai titik tujuan. Kedua cost ini akan menghasilkan sebuah fungsi evaluasi untuk setiap simpul hidup yang dinotasikan seperti dibawah ini :

$$f(n) = g(n) + h(n)$$



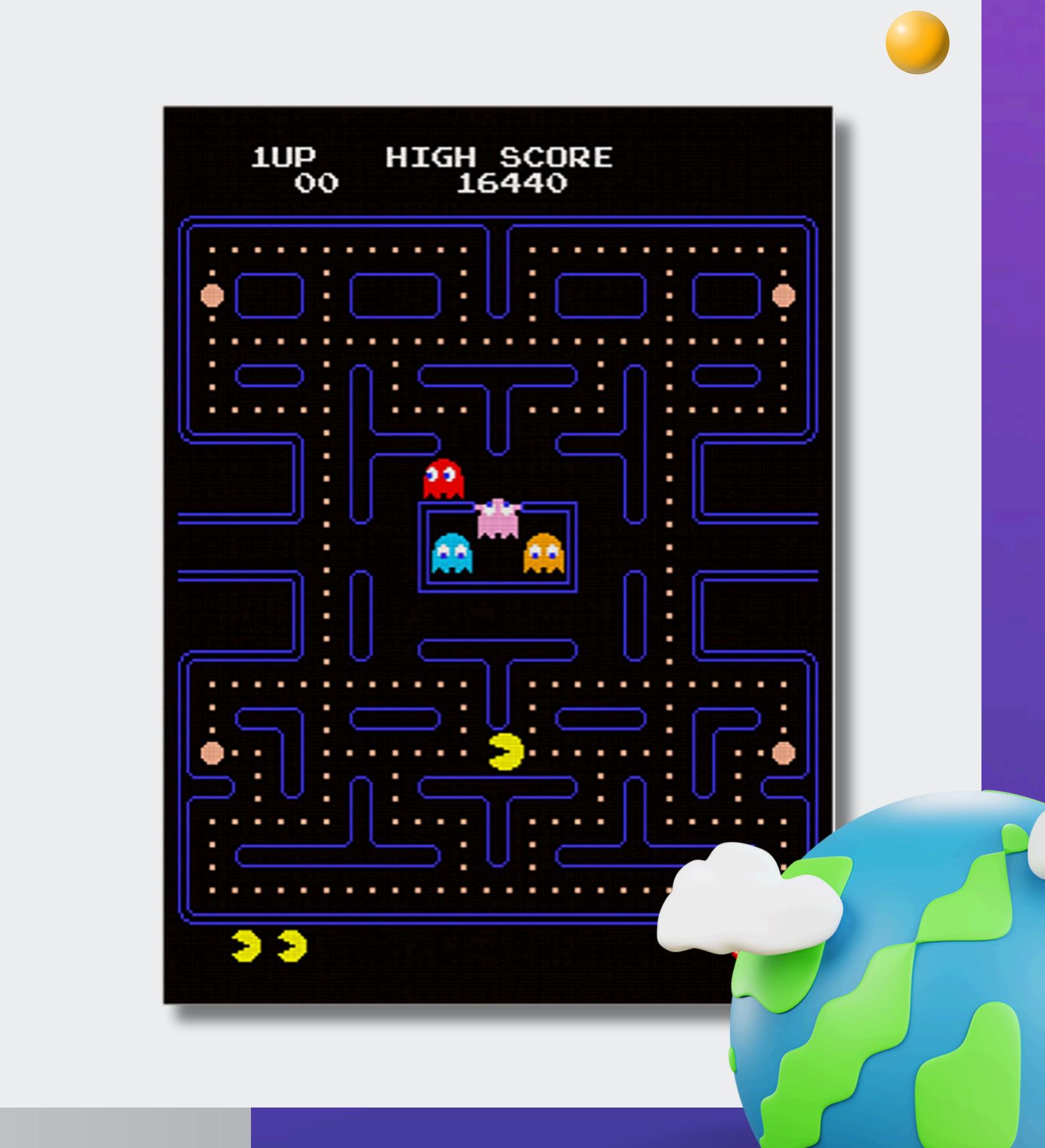
Algoritma A*

Tujuan dari algoritma ini adalah mencari minimum cost untuk mencapai tujuan. Di setiap langkah, sungsi evaluasi akan dihitung untuk setiap simpul hidup. Lalu, untuk menentikan titik selanjutnya yang akan diekspan, simpul hidup akan diurutkan berdasarkan fungsi evaluasi. Simpul yang akan diekspan adalah simpul yang ada di antrian paling depan simpul hidup. Pencarian akan dihentikan jika simpul yang diekspan adalah simpul tujuan.



Game Pacman

Game Pacman merupakan permainan arcade yang pertama kali dirilis di Jepang pada 22 Mei 1980. Permainan pacman merupakan permainan bertingkat / ber level, jika pemain berhasil memakan seluruh titik yang ada di labirin maka pemain akan naik ke level berikutnya. Selama permainan kita akan dihadang oleh musuh yang biasa disebut dengan ghost/hantu, para ghost akan mengejar dan berusaha menangkap kita. Jika kita tertangkap maka nyawa yang kita miliki akan berkurang



Karakter Game

Pada permainan pacman terdapat sejumlah karakter yang memiliki peran masing-masing dalam permainan ini



Pac Man

Merupakan main character pada permainan pacman yang bertugas mengumpulkan titik-titik makanan (disebut "dots" atau "pellets"), menghindari kontak langsung dengan ghost, memakan energi (disebut "power pellets") yang dapat memberikan kekuatan sementara untuk mengejar dan memakan para ghost



Karakter Game



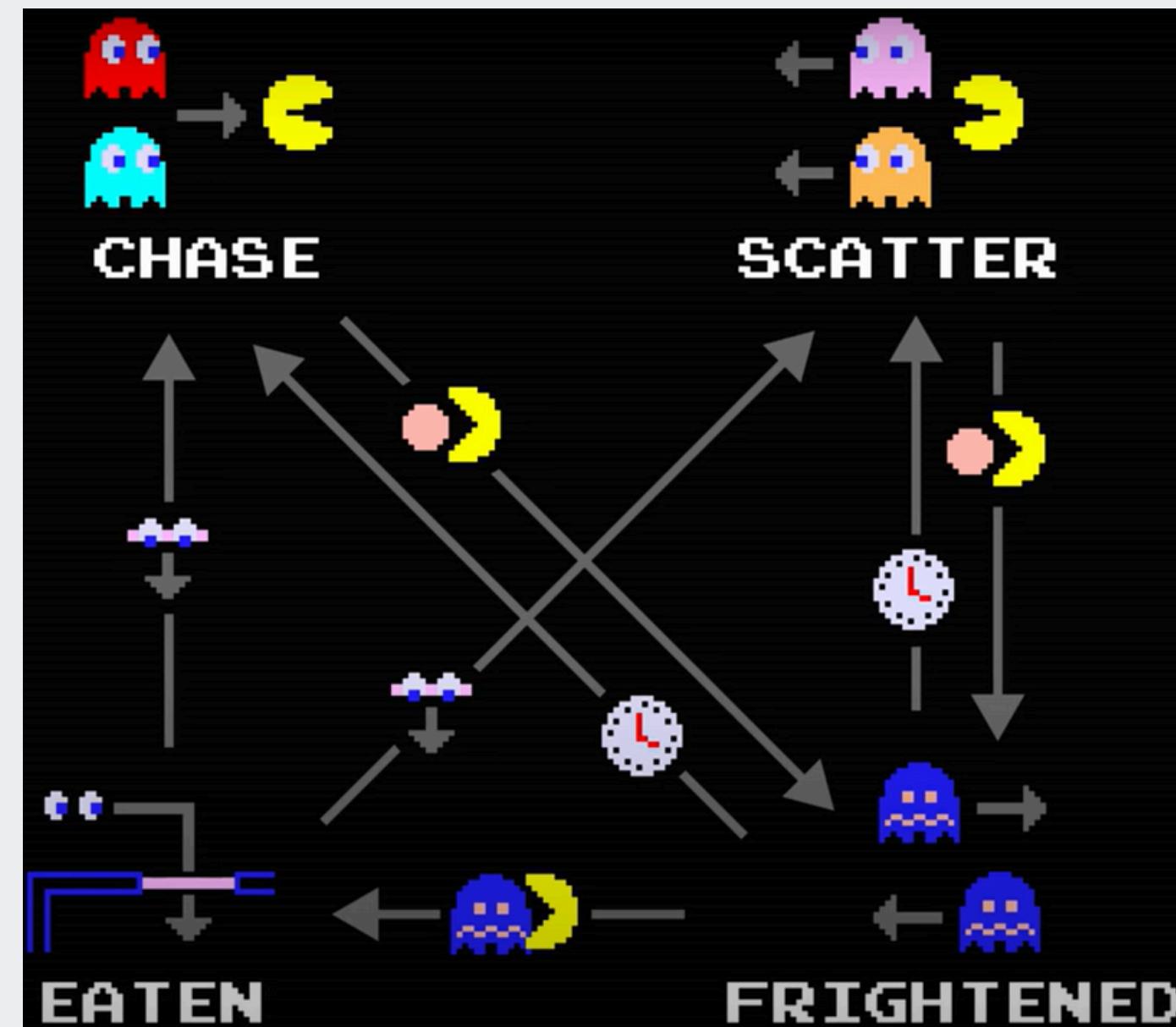
Ghost / Hantu

Merupakan musuh utama dalam permainan Pacman yang berusaha untuk menangkap dan menghalangi pac-man dalam mengumpulkan dot makanan di labirin. Dalam beberapa varian permainan pacman terdapat berbagai jenis ghost yang memiliki perilaku dan kemampuan yang berbeda, seperti kecepatan gerakan, pola pengejaran, dan kemampuan khusus lainnya

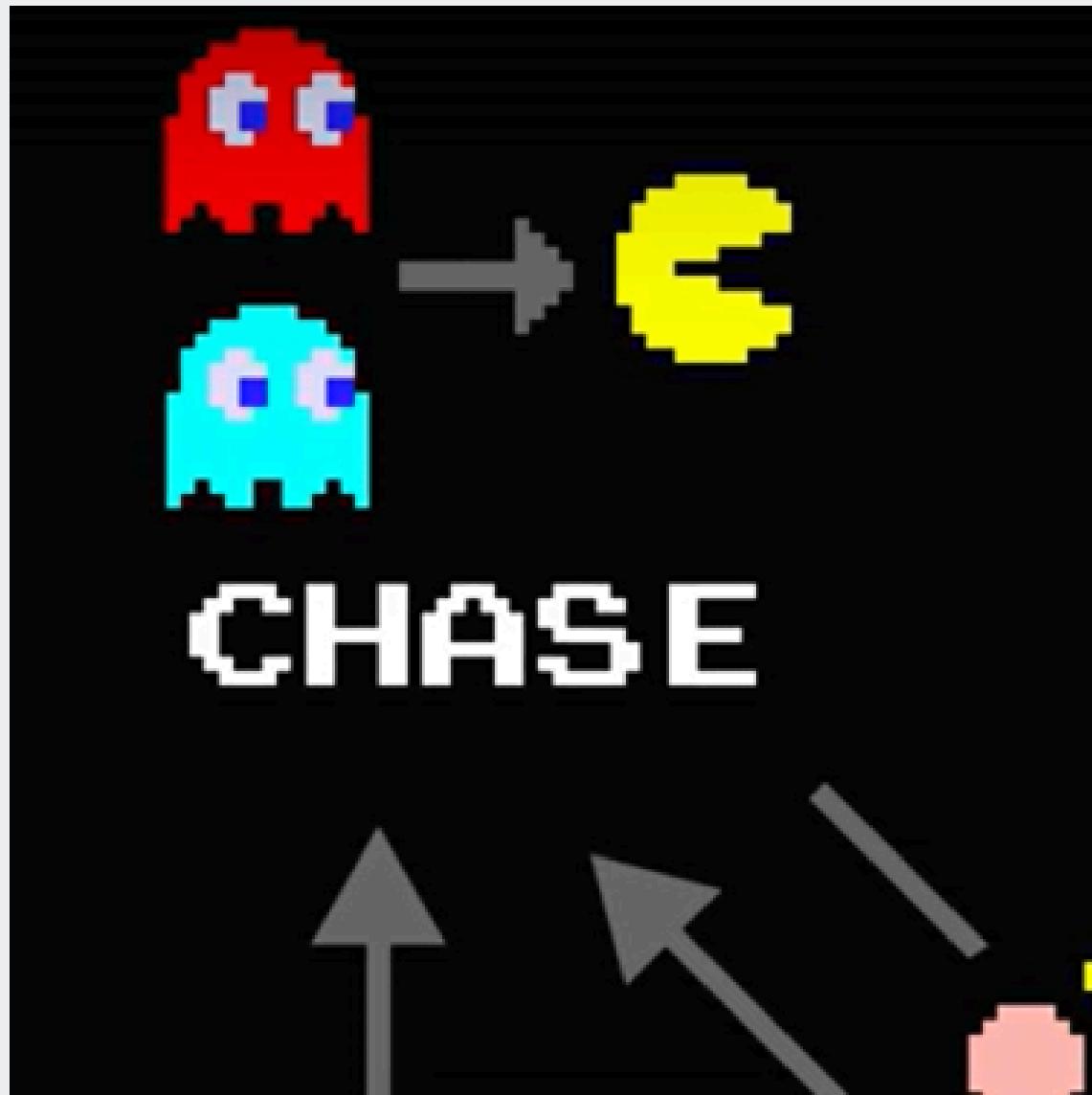


Pergerakan Ghost

Ghost pada permainan pacman memiliki beberapa jenis gerakan untuk setiap peran. Peran yang ada yaitu Blinky, Pink, Inky, dan Clyde. Serta 4 jenis gerakannya yaitu chase, scatter, frightened, dan eaten



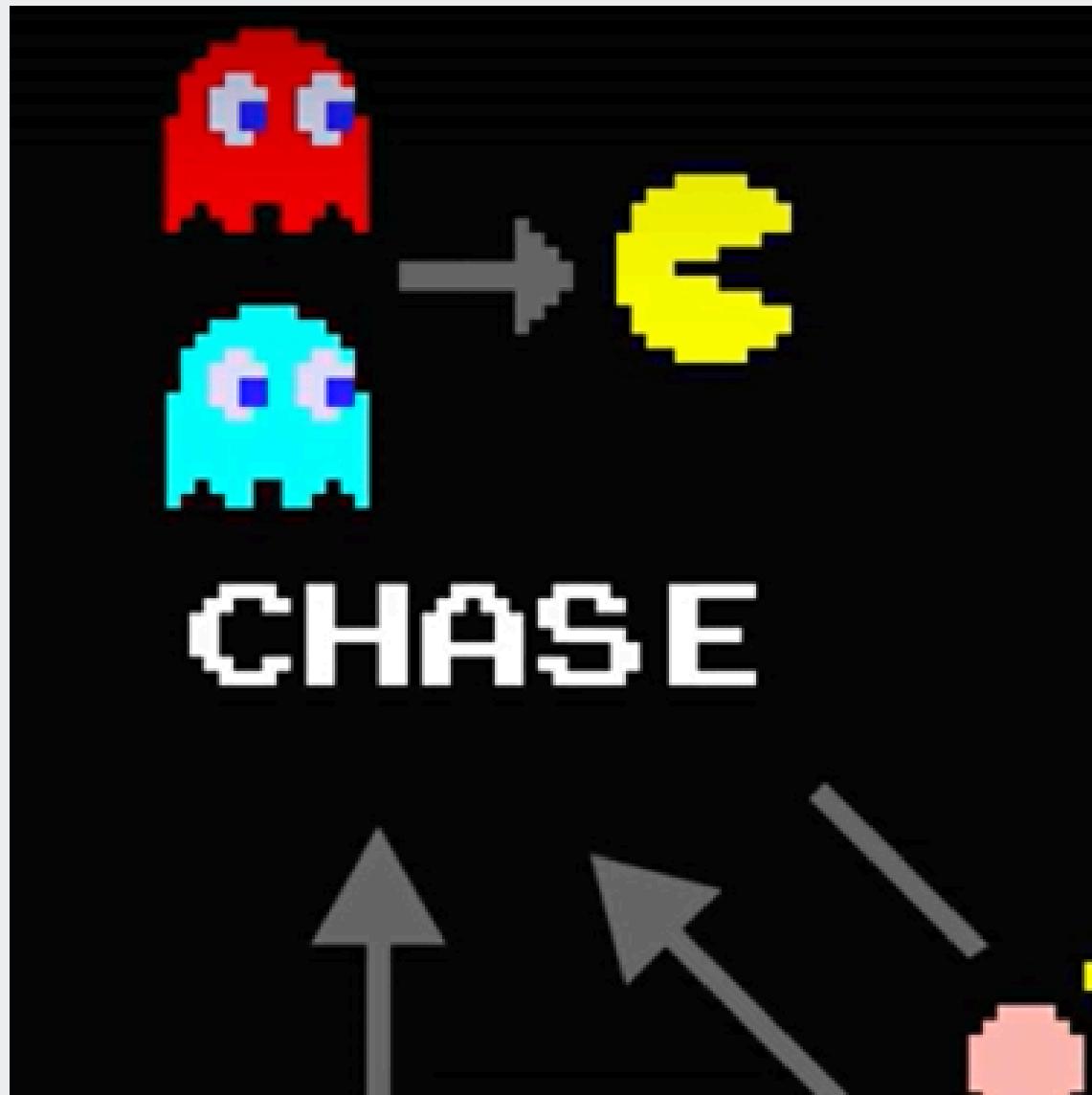
Read more



1. Chase

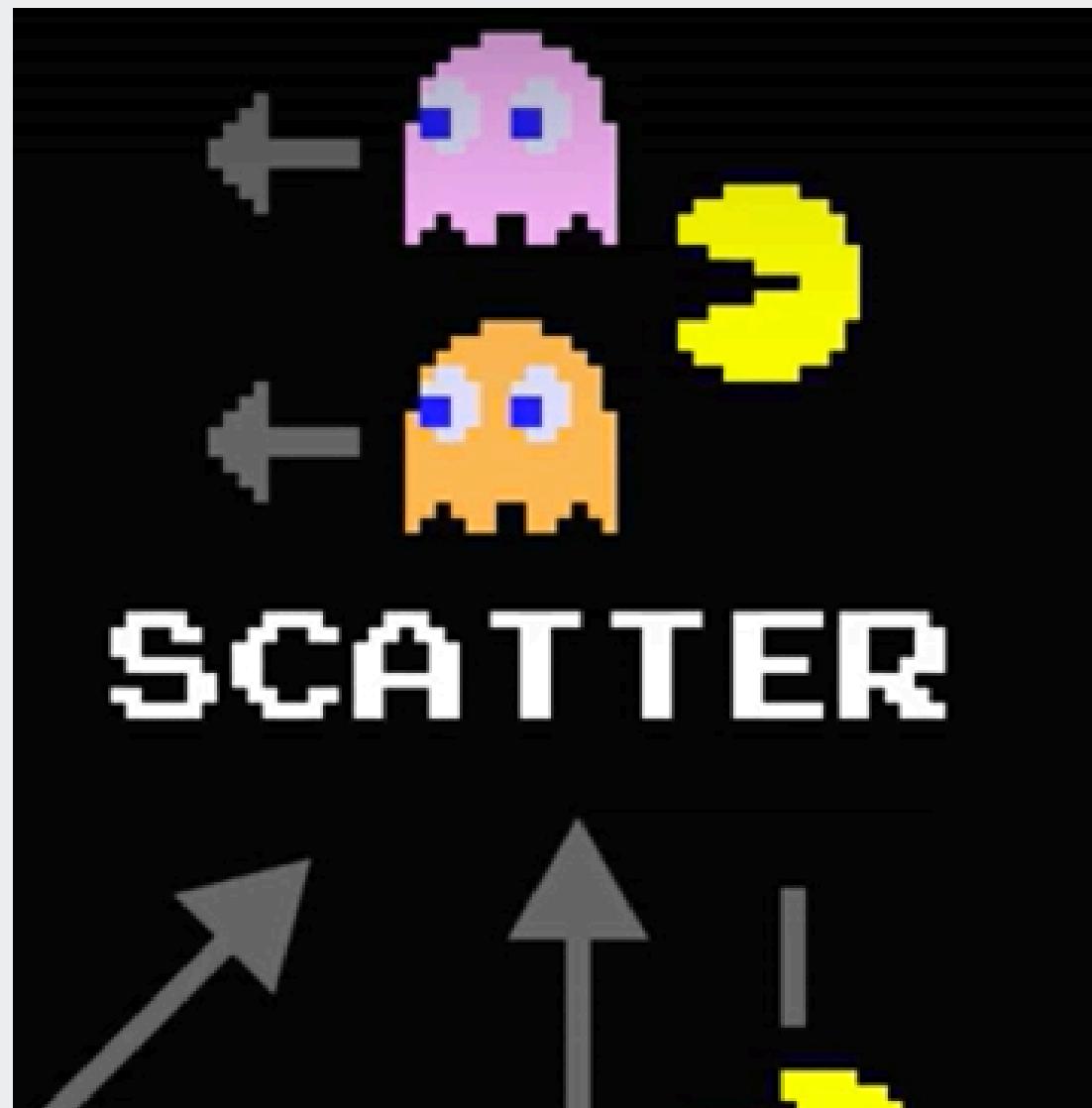
gerakan chase terjadi ketika permainan baru dimulai, para ghost akan keluar dari tempatnya dan mulai mengejar pac man satu per satu. Setiap peran ghost memiliki sifat dan keunikannya masing-masing. Blinky (ghost merah) akan langsung mencari dari pac man dan mengejar dengan agresif

[Read more](#)



Pinky (ghost pink) akan mencoba untuk menyergap pac man dari depan dan menghalangi jalan, Inky (ghost biru neon) akan ber keliling dengan pergerakan yang sulit diprediksi, dan yang terakhir clyde (ghost oranye) akan bergerak secara random dan terlihat menjauhi pac man.

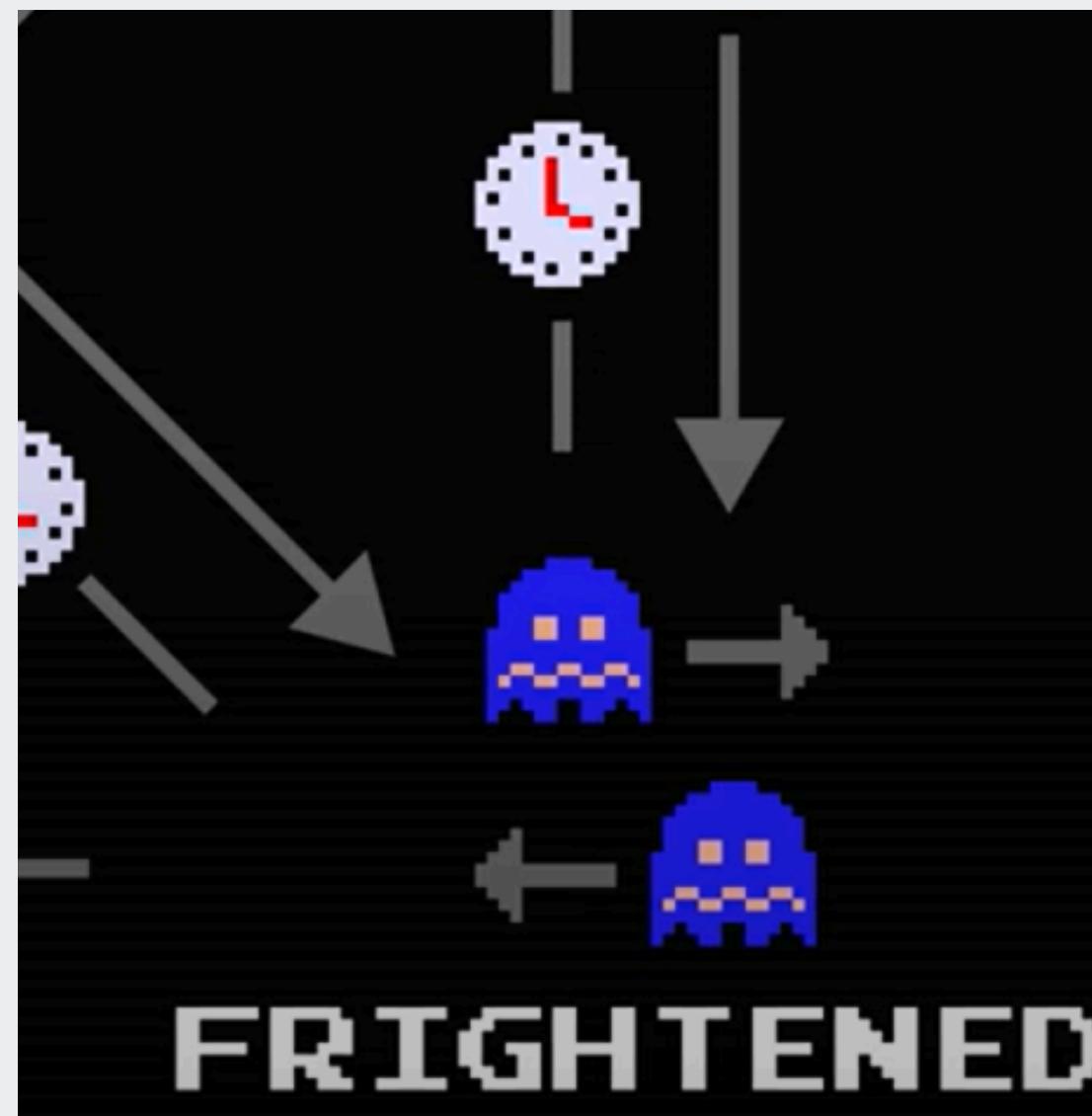
[Read more](#)



2. Scatter

gerakan scatter terjadi dimana ghost akan berhenti mengejar pac man dan akan bergerak secara berulang-ulang pada keempat sudut perta. Blinky akan berputar ke bagian atas kanan, Pinky akan berputar ke area kiri atas, Inky akan berputar ke bawah kanan, dan Clyde akan bergerak ke kiri bawah

[Read more](#)

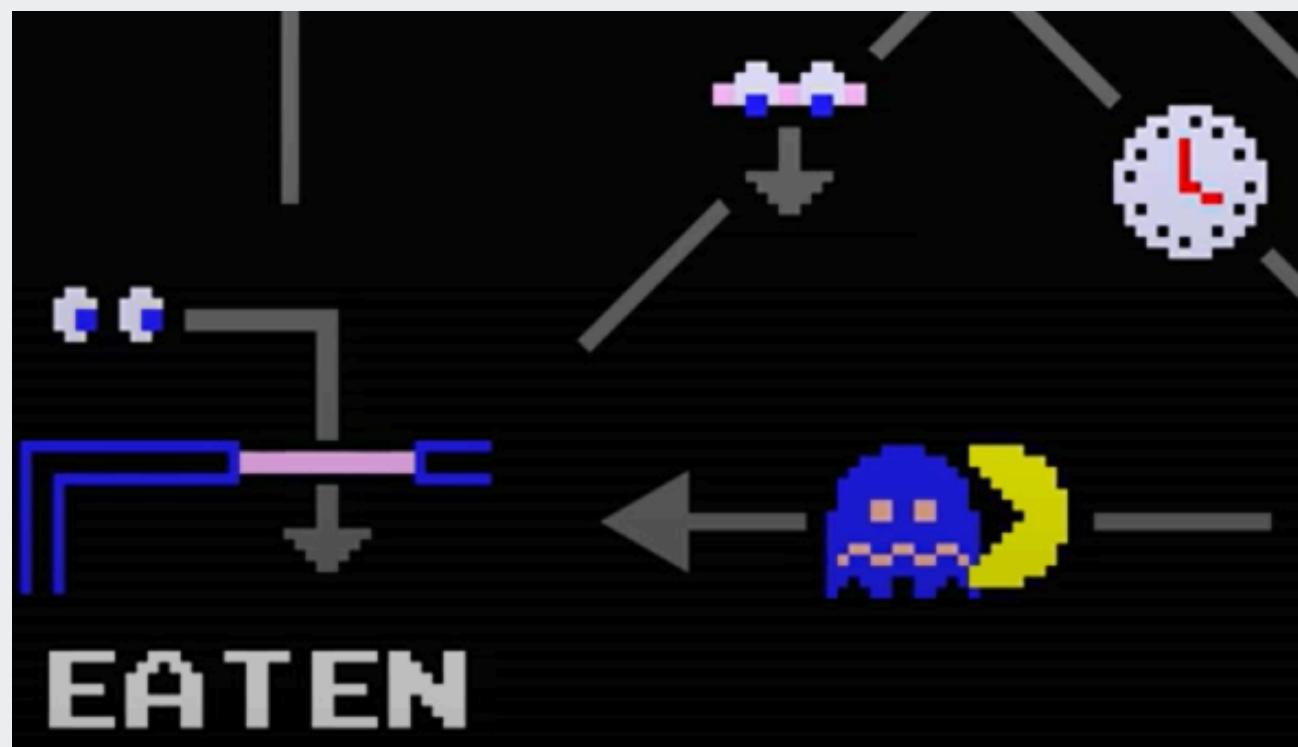


3. Frightened

gerakan ini terjadi ketika pac man berhasil memakan energizer. Saat ini terjadi, maka semua ghost akan berubah warna menjadi biru, dan akan bergerak mengelilingi labirin dan pada saat ini juga pac man dapat mengejar para ghost untuk mendapatkan poin, gerakan ini bersifat sementara

[Read more](#)

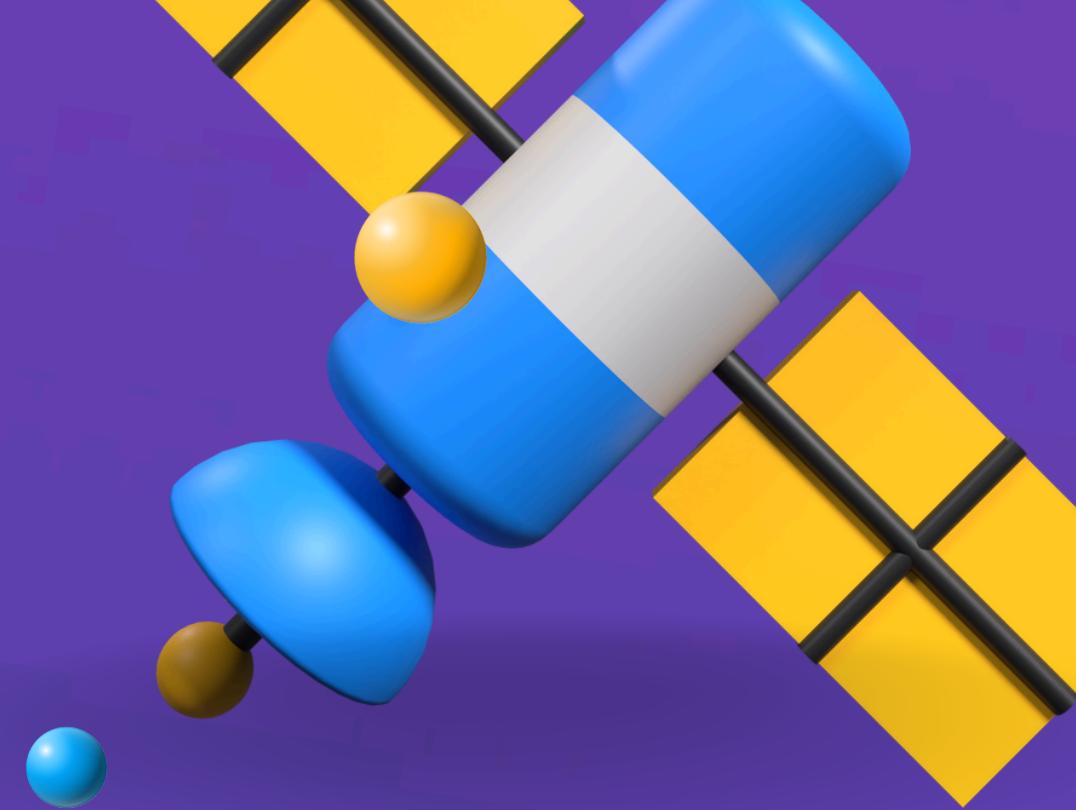
4. Eaten



gerakan ini terjadi ketika pacman berhasil menangkap ghost dalam keadaan frightened, dan ghost harus kembali ke tempat asalnya untuk bisa kembali hidup. Pada tahap ini, ghost akan bergerak secepat mungkin agar bisa kembali ke bentuk asalnya dan kembali mengejar pacman.

Read more

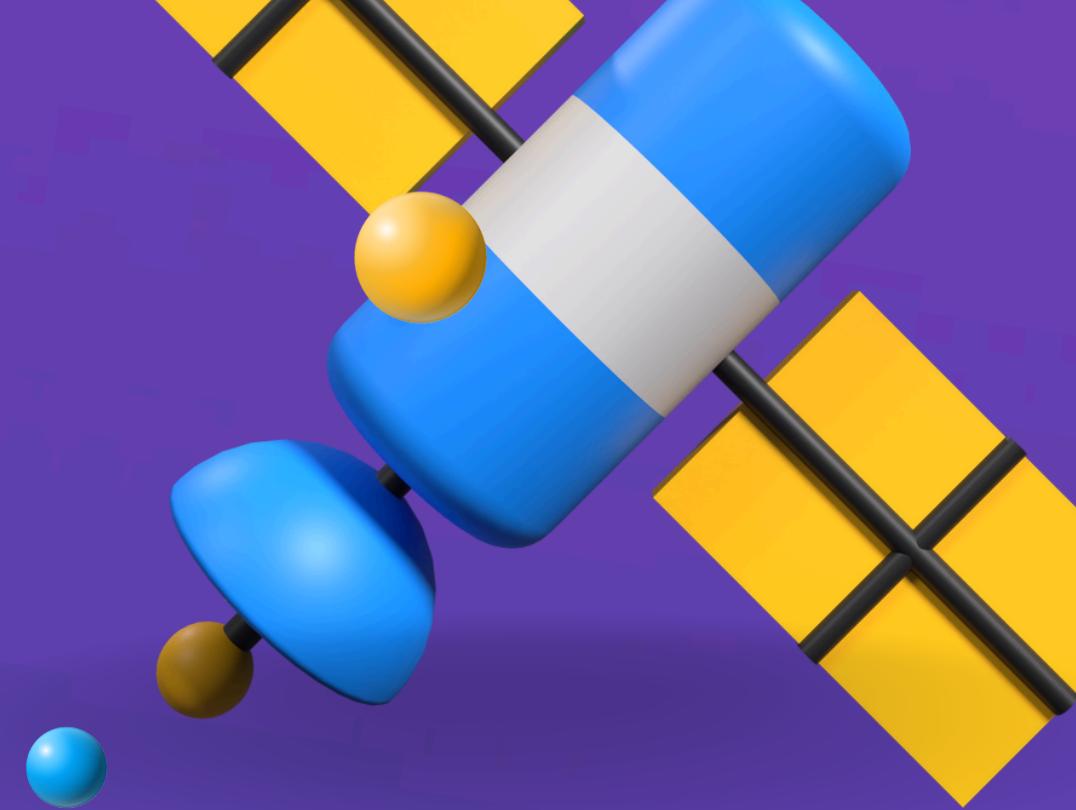
Implementasi Algoritma



Algoritma ini dimulai dengan menentukan posisi Pac-Man dan posisi ghost dalam peta yang direpresentasikan dengan sumbu-X dan sumbu-Y. Langkah selanjutnya adalah mencari fungsi evaluasi $f(n)$ untuk setiap simpul hidup dalam pencarian jalur. Fungsi evaluasi ini terdiri dari dua komponen:

1. Geographical Cost [$g(n)$] : Jumlah biaya geografis dari simpul awal (yaitu, posisi ghost sebelumnya) ke simpul yang sedang dievaluasi. Ini mencerminkan seberapa jauh ghost telah bergerak sejak langkah sebelumnya.
2. Heuristic Cost [$h(n)$] : Estimasi biaya heuristik dari simpul yang dievaluasi ke tujuan akhir, yaitu posisi Pac-Man. Dalam konteks ini, jarak Euclidean digunakan sebagai heuristik, yang mengukur jarak langsung antara dua titik dalam ruang dua dimensi.

Implementasi Algoritma



Titik yang akan dievaluasi fungsi evaluasinya adalah tetangga dari ghost, yaitu titik-titik di atas, bawah, kanan, dan kiri ghost. Ini karena ghost hanya dapat bergerak ke arah-arah tersebut dan tidak bisa bergerak secara diagonal. Jika dua atau lebih titik tetangga memiliki nilai fungsi evaluasi yang sama, urutan prioritas dalam ekspansi simpul adalah titik di atas, di kiri, di bawah, dan di kanan.

Dengan menggunakan informasi ini, algoritma akan mencari jalur terpendek dari ghost ke posisi Pac-Man, dengan mempertimbangkan biaya geografis dan estimasi heuristik untuk setiap simpul yang dievaluasi. Ini memungkinkan ghost untuk bergerak secara cerdas menuju Pac-Man dengan mempertimbangkan kedekatan dan kecocokan dengan posisi target.



Dokumentasi

```
1 import pygame
2 import random
3 import math
4 from queue import PriorityQueue
5
6 # Inisialisasi warna
7 BLACK = (0, 0, 0)
8 WHITE = (255, 255, 255)
9 YELLOW = (255, 255, 0)
10 RED = (255, 0, 0)
11 BLUE = (0, 0, 255)
12 |
13 # Inisialisasi ukuran layar dan grid
14 CELL_SIZE = 20
15 GRID_WIDTH = 28
16 GRID_HEIGHT = 31
17 MARGIN = 1
18 SCREEN_WIDTH = GRID_WIDTH * CELL_SIZE + (GRID_WIDTH + 1) * MARGIN
19 SCREEN_HEIGHT = GRID_HEIGHT * CELL_SIZE + (GRID_HEIGHT + 1) * MARGIN
20
21 # Inisialisasi arah pergerakan
22 UP = (0, -1)
23 DOWN = (0, 1)
24 LEFT = (-1, 0)
25 RIGHT = (1, 0)
26
27 # Inisialisasi kecepatan gerakan hantu
28 GHOST_SPEED = 0.05
29
```

```
# Inisialisasi kelas node
class Node:
    def __init__(self, row, col):
        self.row = row
        self.col = col
        self.wall = False

# Inisialisasi kelas Pacman
class Pacman:
    def __init__(self, row, col):
        self.row = row
        self.col = col
        self.score = 0

    def move(self, direction, grid):
        next_row = self.row + direction[1]
        next_col = self.col + direction[0]

        # Periksa apakah sel yang akan dituju bukan dinding
        if 0 <= next_row < GRID_HEIGHT and 0 <= next_col < GRID_WIDTH and not grid[next_row][next_col].wall:
            self.row = next_row
            self.col = next_col
```

Dokumentasi

```
# Inisialisasi kelas Ghost
class Ghost:
    def __init__(self, row, col):
        self.row = row
        self.col = col
        self.path = []
        self.direction = random.choice([UP, DOWN, LEFT, RIGHT]) # Inisialisasi arah acak

    def heuristic(self, a, b):
        return abs(a[0] - b[0]) + abs(a[1] - b[1])

    def find_path(self, grid, target):
        open_list = PriorityQueue()
        open_list.put((0, (self.row, self.col)))
        came_from = {}
        cost_so_far = {(self.row, self.col): 0}

        while not open_list.empty():
            current_cost, current_node = open_list.get()

            if current_node == target:
                self.path = self.reconstruct_path(came_from, current_node)
                return

            for direction in [UP, DOWN, LEFT, RIGHT]:
                next_row = current_node[0] + direction[1]
                next_col = current_node[1] + direction[0]

                if not (0 <= next_row < GRID_HEIGHT and 0 <= next_col < GRID_WIDTH):
```

```
if not (0 <= next_row < GRID_HEIGHT and 0 <= next_col < GRID_WIDTH):
    continue

neighbor = (next_row, next_col)
new_cost = cost_so_far[current_node] + 1

if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:
    cost_so_far[neighbor] = new_cost
    priority = new_cost + self.heuristic(target, neighbor)
    open_list.put((priority, neighbor))
    came_from[neighbor] = current_node

def reconstruct_path(self, came_from, current_node):
    path = []
    while current_node in came_from:
        path.append(current_node)
        current_node = came_from[current_node]
    path.reverse()
    return path

def move(self, grid):
    if random.random() < GHOST_SPEED:
        # Perbarui arah jika hantu sudah mencapai titik tujuan dalam jalur
        if not self.path or (self.row, self.col) == self.path[-1]:
            self.direction = random.choice([UP, DOWN, LEFT, RIGHT])
            self.find_path(grid, (random.randint(0, GRID_HEIGHT - 1), random.randint(0, GRID_WIDTH - 1)))
```

Dokumentasi

```
def move(self, grid):
    if random.random() < GHOST_SPEED:
        # Perbarui arah jika hantu sudah mencapai titik tujuan dalam jalur
        if not self.path or (self.row, self.col) == self.path[-1]:
            self.direction = random.choice([UP, DOWN, LEFT, RIGHT])
            self.find_path(grid, (random.randint(0, GRID_HEIGHT - 1), random.randint(0, GRID_WIDTH - 1)))

        next_row = self.row + self.direction[1]
        next_col = self.col + self.direction[0]

        # Periksa apakah sel yang akan dituju bukan dinding
        if 0 <= next_row < GRID_HEIGHT and 0 <= next_col < GRID_WIDTH and not grid[next_row][next_col].wall:
            self.row = next_row
            self.col = next_col
        else:
            # Jika bertabrakan dengan dinding, cari arah lain yang kosong
            possible_directions = [UP, DOWN, LEFT, RIGHT]
            random.shuffle(possible_directions)
            for direction in possible_directions:
                new_row = self.row + direction[1]
                new_col = self.col + direction[0]
                if 0 <= new_row < GRID_HEIGHT and 0 <= new_col < GRID_WIDTH and not grid[new_row][new_col].wall:
                    self.row = new_row
                    self.col = new_col
                    self.direction = direction
                    break
```

```
# Fungsi untuk menggambar grid
def draw_grid(grid, screen):
    for row in range(GRID_HEIGHT):
        for col in range(GRID_WIDTH):
            color = BLACK
            if grid[row][col].wall:
                color = BLUE
            pygame.draw.rect(screen, color, [(MARGIN + CELL_SIZE) * col + MARGIN,
                                             (MARGIN + CELL_SIZE) * row + MARGIN,
                                             CELL_SIZE,
                                             CELL_SIZE])

# Fungsi untuk membuat labirin
def create_maze():
    maze = [[Node(row, col) for col in range(GRID_WIDTH)] for row in range(GRID_HEIGHT)]

    # Inisialisasi semua sel sebagai dinding
    for row in range(GRID_HEIGHT):
        for col in range(GRID_WIDTH):
            maze[row][col].wall = True

    # Mulai dari sel tengah dan lakukan rekursi untuk membuka jalan
    start_row, start_col = GRID_HEIGHT // 2, GRID_WIDTH // 2
    maze[start_row][start_col].wall = False
    stack = [(start_row, start_col)]

    while stack:
        current_row, current_col = stack[-1]
        neighbors = [(current_row + 2, current_col), (current_row - 2, current_col),
```

Dokumentasi

```
while stack:
    current_row, current_col = stack[-1]
    neighbors = [(current_row + 2, current_col), (current_row - 2, current_col),
                  (current_row, current_col + 2), (current_row, current_col - 2)]
    unvisited_neighbors = [(row, col) for row, col in neighbors if 0 <= row < GRID_HEIGHT and 0 <= col < GRID_WIDTH and maze[row][col].wall == False]
    if unvisited_neighbors:
        next_row, next_col = random.choice(unvisited_neighbors)
        maze[next_row][next_col].wall = False
        maze[(current_row + next_row) // 2][(current_col + next_col) // 2].wall = False
        stack.append((next_row, next_col))
    else:
        stack.pop()

return maze

# Fungsi untuk menentukan posisi awal hantu dengan jarak yang cukup jauh dari Pac-Man
def generate_ghost_positions(pacman_row, pacman_col):
    max_distance = 10 # Jarak minimal antara Pac-Man dan hantu
    ghost_positions = []

    while len(ghost_positions) < 3: # Ubah angka 3 sesuai dengan jumlah hantu
        ghost_row = random.randint(0, GRID_HEIGHT - 1)
        ghost_col = random.randint(0, GRID_WIDTH - 1)
        # Hitung jarak Euclidean antara Pac-Man dan hantu
        distance = math.sqrt((ghost_row - pacman_row) ** 2 + (ghost_col - pacman_col) ** 2)
        if distance >= max_distance:
            ghost_positions.append((ghost_row, ghost_col))
```

Dokumentasi

```
def generate_ghost_positions(pacman_row, pacman_col):
    return ghost_positions

# Fungsi utama
def main():
    pygame.init()

    # Inisialisasi layar
    screen = pygame.display.set_mode([SCREEN_WIDTH, SCREEN_HEIGHT])
    pygame.display.set_caption("Pac-Man")

    # Inisialisasi labirin
    grid = create_maze()

    # Inisialisasi karakter Pac-Man
    pacman = Pacman(13, 15)

    # Inisialisasi hantu
    ghost_positions = generate_ghost_positions(pacman.row, pacman.col)
    ghosts = [Ghost(row, col) for row, col in ghost_positions]

    # Inisialisasi umpan
    food_positions = [(row, col) for row in range(GRID_HEIGHT) for col in range(GRID_WIDTH)
                      if not grid[row][col].wall and (row, col) != (pacman.row, pacman.col)]
    for ghost in ghosts:
        food_positions = [(row, col) for (row, col) in food_positions if (row, col) != (ghost.row, ghost.col)]
    food_positions = random.sample(food_positions, len(food_positions) // 2) # Ambil setengah dari posisi makanan
    foods = food_positions
```

Dokumentasi

```
# Main loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                pacman.move(UP, grid)
            elif event.key == pygame.K_DOWN:
                pacman.move(DOWN, grid)
            elif event.key == pygame.K_LEFT:
                pacman.move(LEFT, grid)
            elif event.key == pygame.K_RIGHT:
                pacman.move(RIGHT, grid)

    # Gerakkan hantu
    for ghost in ghosts:
        ghost.move(grid)

    # Hapus makanan yang dimakan oleh Pac-Man
    if (pacman.row, pacman.col) in food_positions:
        food_positions.remove((pacman.row, pacman.col))
        pacman.score += 10

    # Cek tabrakan antara Pac-Man dan hantu
    for ghost in ghosts:
        if (pacman.row, pacman.col) == (ghost.row, ghost.col):
            print("Game Over! Your score:", pacman.score)
            pygame.quit()
            return
```

```
def main():
    for ghost in ghosts:
        if (pacman.row, pacman.col) == (ghost.row, ghost.col):
            print("Game Over! Your score:", pacman.score)
            pygame.quit()
            return

    # Clear the screen and draw the grid
    screen.fill(BLACK) # Ubah latar belakang layar menjadi hitam
    draw_grid(grid, screen)

    # Gambar makanan
    for food in foods:
        pygame.draw.circle(screen, WHITE, ((MARGIN + CELL_SIZE) * food[1] + MARGIN + CELL_SIZE // 2,
                                         (MARGIN + CELL_SIZE) * food[0] + MARGIN + CELL_SIZE // 2), CELL_SIZE // 6)

    # Gambar Pac-Man
    pygame.draw.circle(screen, YELLOW, ((MARGIN + CELL_SIZE) * pacman.col + MARGIN + CELL_SIZE // 2,
                                         (MARGIN + CELL_SIZE) * pacman.row + MARGIN + CELL_SIZE // 2), CELL_SIZE // 2)

    # Gambar hantu
    for ghost in ghosts:
        pygame.draw.circle(screen, RED, ((MARGIN + CELL_SIZE) * ghost.col + MARGIN + CELL_SIZE // 2,
                                         (MARGIN + CELL_SIZE) * ghost.row + MARGIN + CELL_SIZE // 2), CELL_SIZE // 2)

    # Update the display
    pygame.display.flip()

    pygame.quit()

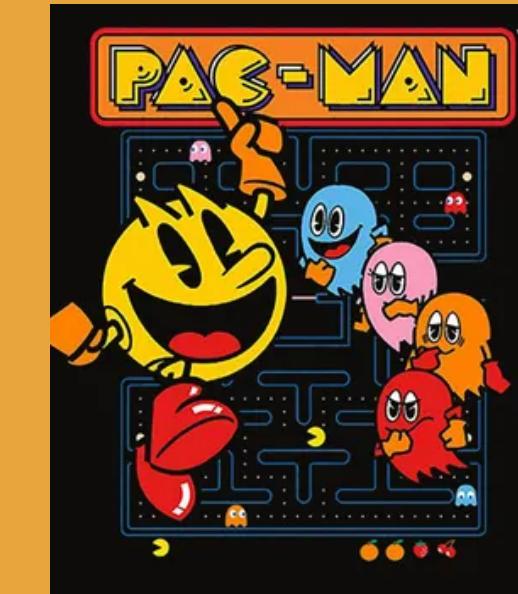
if __name__ == "__main__":
    main()
```

Dokumentasi



Daftar Pustaka

- Munir, Rinaldi. "Strategi Algoritma Pencarian untuk Permainan Catur." Seminar Teknologi Informasi dan Multimedia (STIMIK), 2020. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Makalah/stima2020k2-036.pdf>



Thank You Everyone

