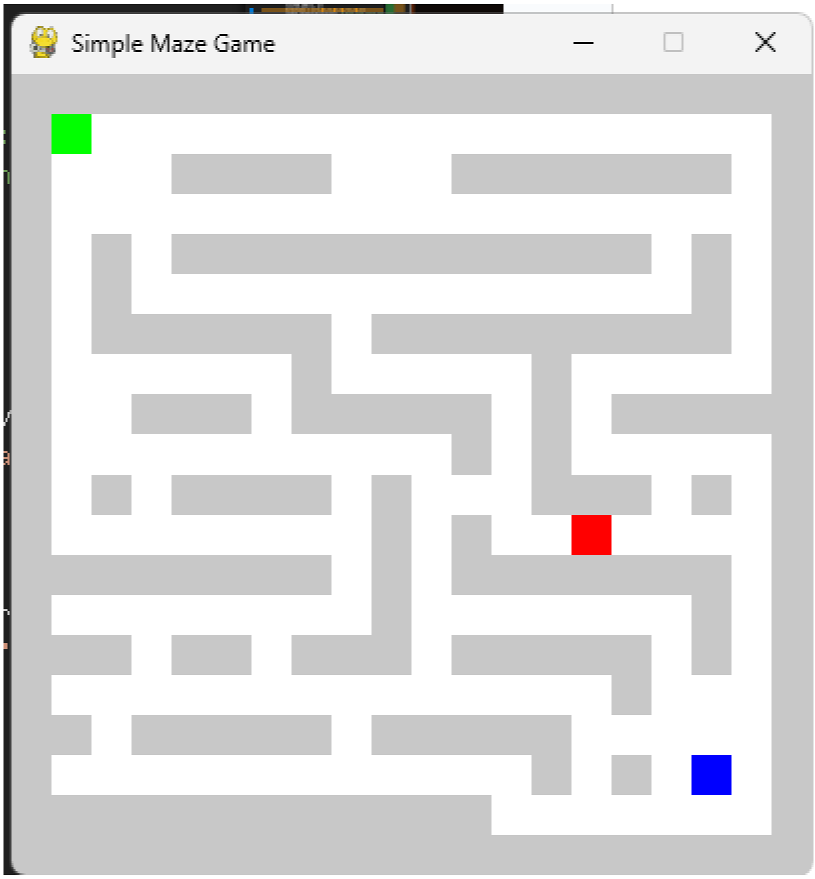
Alfadito Aulia Denova
5025211157
DAA (K)
Quiz 2

Github Repository
https://github.com/AlfaDitoOnGithub/Python.git

Summary and Overview
The Question is to create a project that implements an algorithm that has been discussed. So, I created a game that uses BFS and DFS for its enemy. Using Python and pygame library, for its visual and graphics.

Designing the Game



The premise of the game is simple. Move the green box (player) to the blue box (goal) whilst avoiding the red box (enemy). This game uses DFS and BFS algorithm for the enemy behaviour with a slight modification to gamify it more.

The Maze itself is constructed by a two-dimension array of 0s and 1s, where 0s are the path and 1s are the walls.

At the start of the game the enemy will use DFS to ‘patrol’ the nearby area, then in 15 second time, adjustable, the AI has a random chance to switch to BFS where the enemy will then be actively chasing the player.

The Implementation

The Usage of BFS and DFS are somewhat easy to implement in this particular case, with a slight variation.

The BFS has no adjustment, but the DFS has one, that is to randomize where to check next. This is done so that the enemy can randomly patrol the area.

In DFS:

```
def dfs(maze, start, end):
    stack = [(start[0], start[1], [])]
    visited = set()

    while stack:
        x, y, path = stack.pop()
        if (x, y) == (end[0], end[1]):
            return path + [(x, y)]
        if (x, y) not in visited:
            visited.add((x, y))
            directions = [(-1,0),(1,0),(0,-1),(0,1)]
      ----> random.shuffle(directions)  # Acak arah untuk patroli
            for dx, dy in directions:
                ndx, ndy = x + dx, y + dy
                if 0 <= ndx < len(maze) and 0 <= ndy < len(maze[0]) and
maze[ndx][ndy] == 0:
                    stack.append((ndx, ndy, path + [(x, y)]))
    return []
```

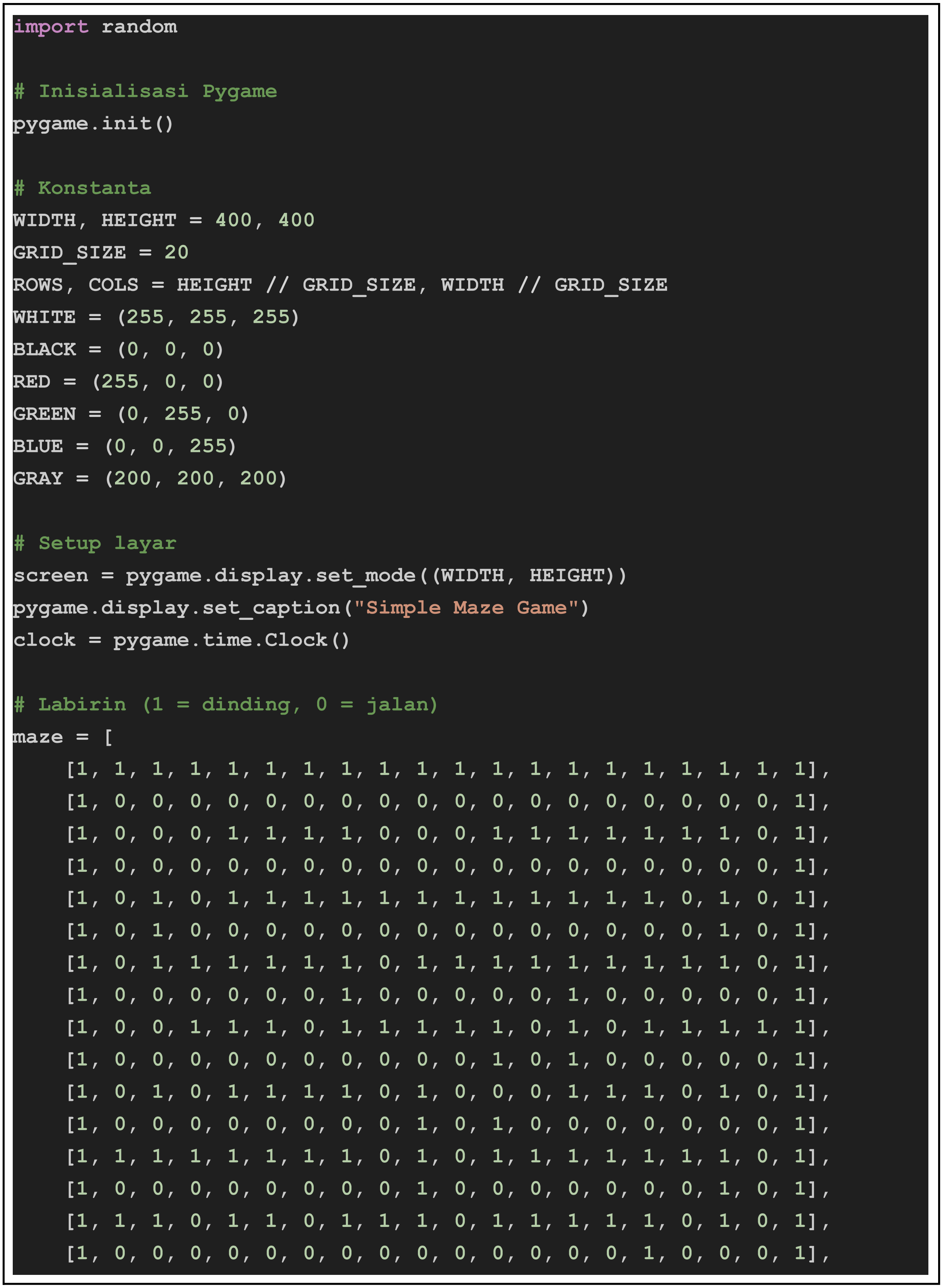
random.shuffle(direction) will shuffle the next direction the algorithm picks.

In BFS:

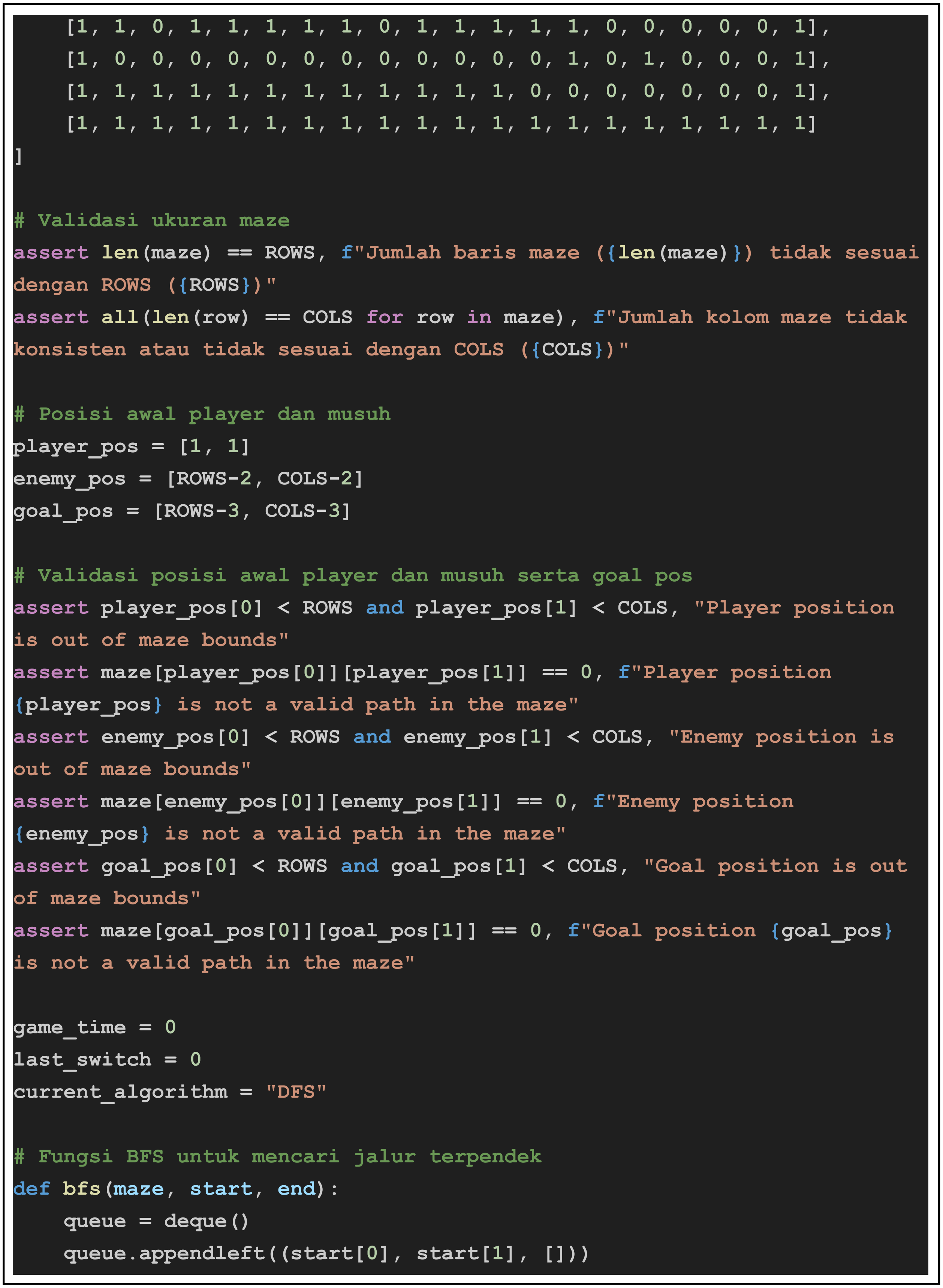
```
def bfs(maze, start, end):
    queue = deque()
    queue.appendleft((start[0], start[1], []))
    visited = set()
    visited.add((start[0], start[1]))

    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Atas, bawah, kiri, kanan

    while queue:
        x, y, path = queue.pop()

        if (x, y) == (end[0], end[1]):
            return path + [(x, y)]

        for dx, dy in directions:
            bx, by = x + dx, y + dy
            if 0 <= bx < ROWS and 0 <= by < COLS and maze[bx][by] == 0 and (bx, by) not in visited:
                visited.add((bx, by))
                queue.appendleft((bx, by, path + [(x, y)]))

    return []  # Jika tidak ditemukan jalur
```

Full Source Code

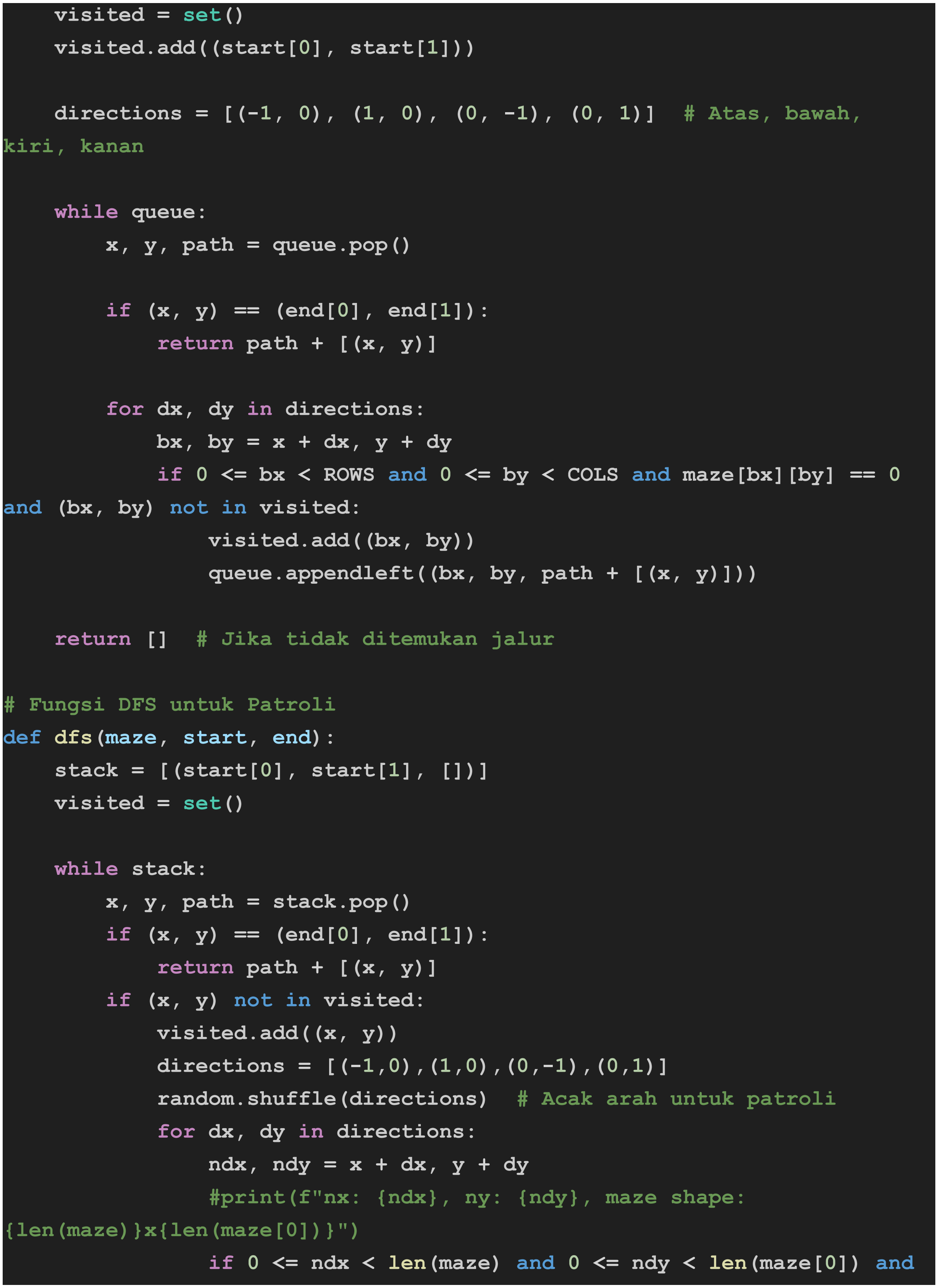
```
import pygame
import sys
from collections import deque
import time
import math
```



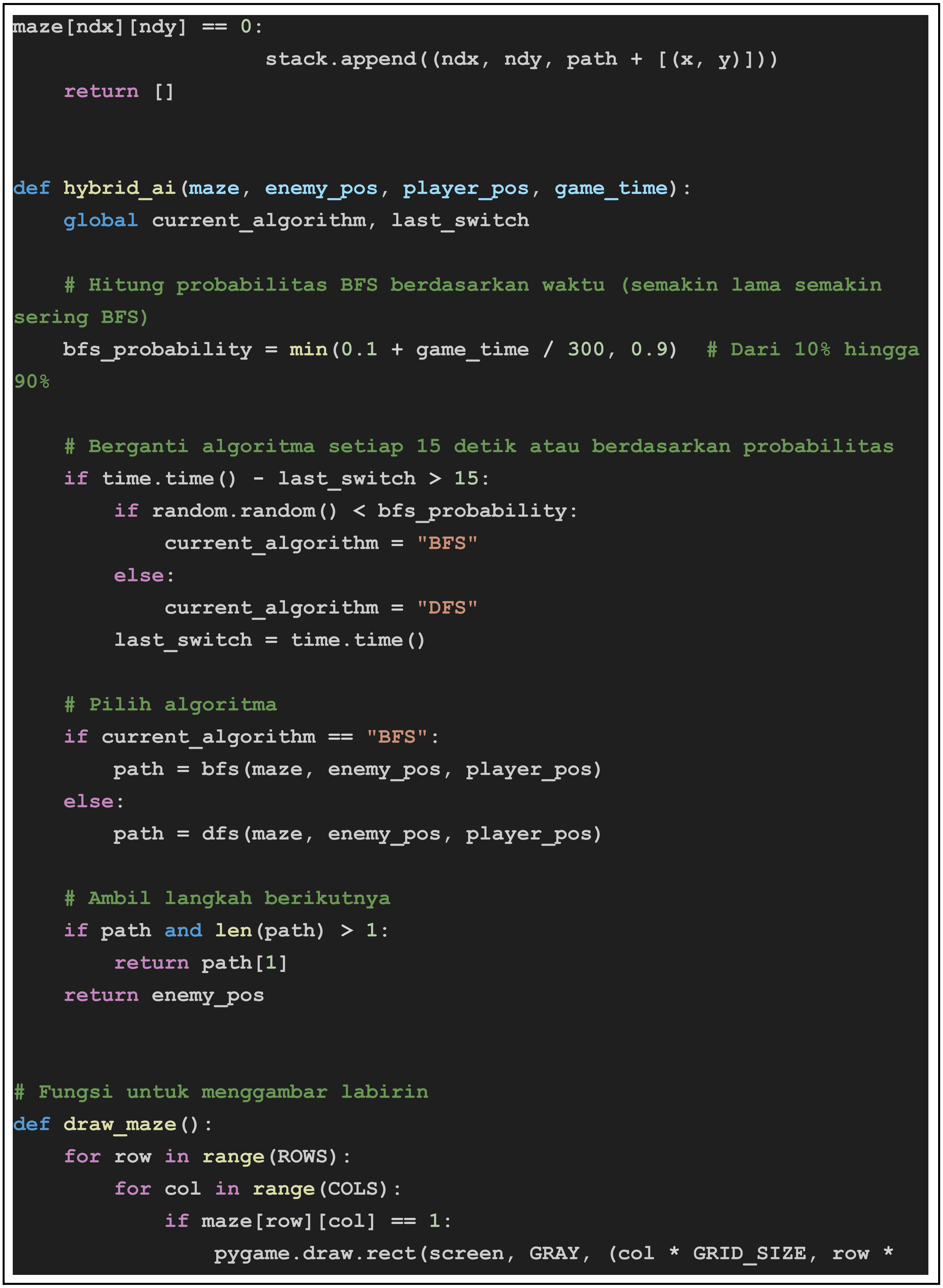
```
import random

# Inisialisasi Pygame
pygame.init()

# Konstanta
WIDTH, HEIGHT = 400, 400
GRID_SIZE = 20
ROWS, COLS = HEIGHT // GRID_SIZE, WIDTH // GRID_SIZE
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
GRAY = (200, 200, 200)

# Setup layar
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Simple Maze Game")
clock = pygame.time.Clock()

# Labirin (1 = dinding, 0 = jalan)
maze = [
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1],
    [1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
```



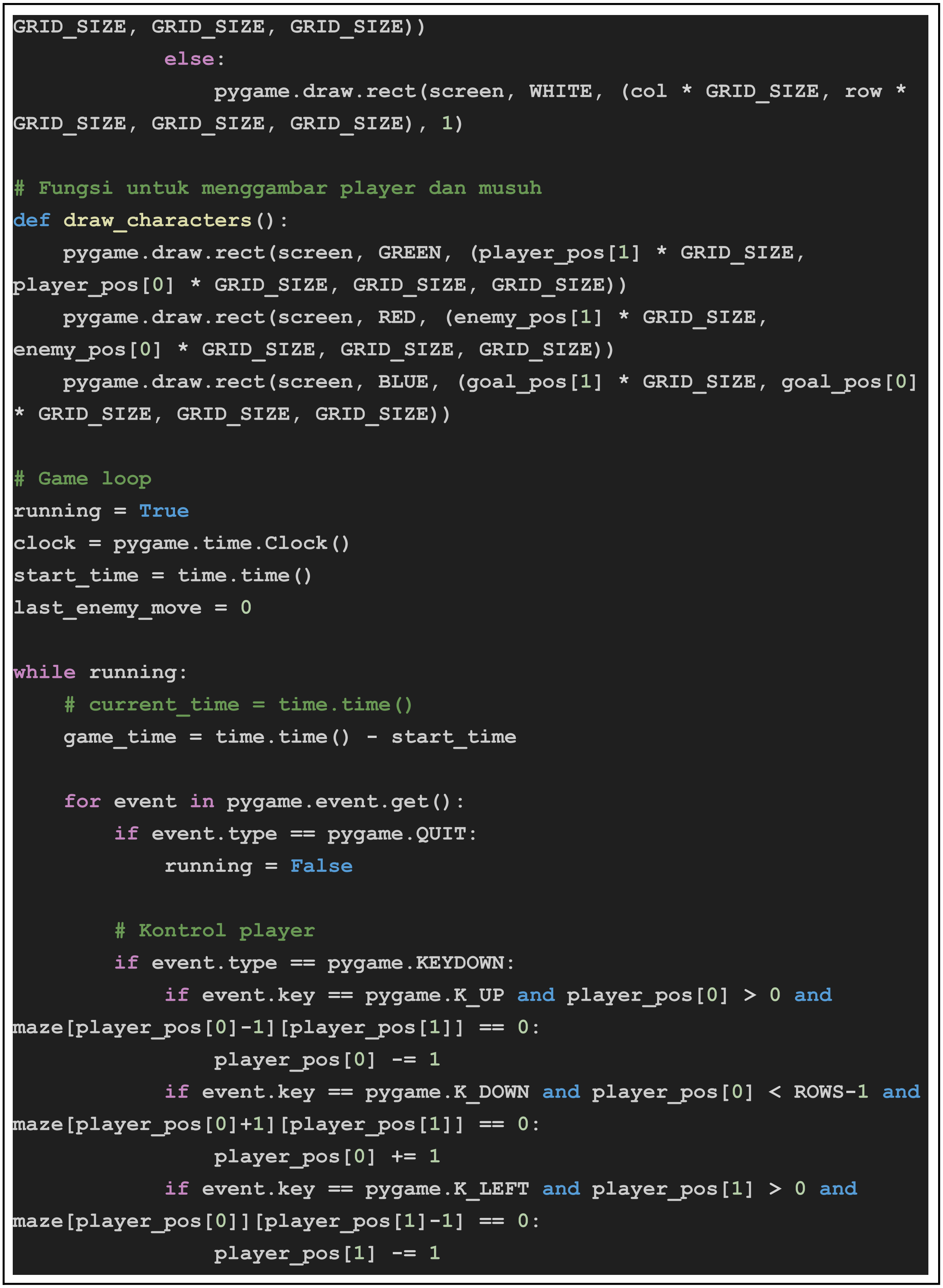
```python
    [1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
]

# Validasi ukuran maze
assert len(maze) == ROWS, f"Jumlah baris maze ({len(maze)}) tidak sesuai dengan ROWS ({ROWS})"
assert all(len(row) == COLS for row in maze), f"Jumlah kolom maze tidak konsisten atau tidak sesuai dengan COLS ({COLS})"

# Posisi awal player dan musuh
player_pos = [1, 1]
enemy_pos = [ROWS-2, COLS-2]
goal_pos = [ROWS-3, COLS-3]

# Validasi posisi awal player dan musuh serta goal pos
assert player_pos[0] < ROWS and player_pos[1] < COLS, "Player position is out of maze bounds"
assert maze[player_pos[0]][player_pos[1]] == 0, f"Player position {player_pos} is not a valid path in the maze"
assert enemy_pos[0] < ROWS and enemy_pos[1] < COLS, "Enemy position is out of maze bounds"
assert maze[enemy_pos[0]][enemy_pos[1]] == 0, f"Enemy position {enemy_pos} is not a valid path in the maze"
assert goal_pos[0] < ROWS and goal_pos[1] < COLS, "Goal position is out of maze bounds"
assert maze[goal_pos[0]][goal_pos[1]] == 0, f"Goal position {goal_pos} is not a valid path in the maze"

game_time = 0
last_switch = 0
current_algorithm = "DFS"

# Fungsi BFS untuk mencari jalur terpendek
def bfs(maze, start, end):
    queue = deque()
    queue.appendleft((start[0], start[1], []))
```



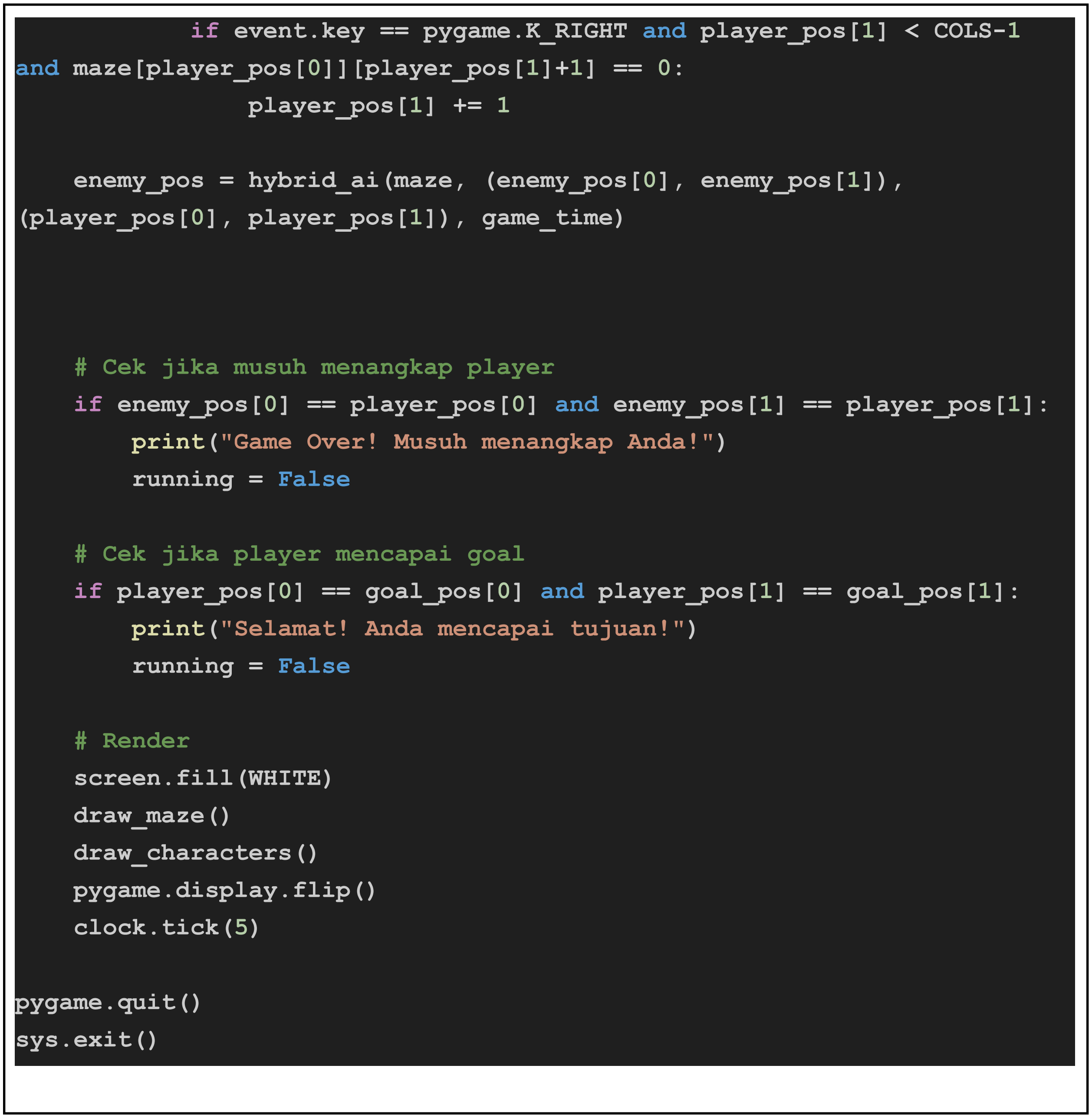
```
    visited = set()
    visited.add((start[0], start[1]))

    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Atas, bawah, kiri, kanan

    while queue:
        x, y, path = queue.pop()

        if (x, y) == (end[0], end[1]):
            return path + [(x, y)]

        for dx, dy in directions:
            bx, by = x + dx, y + dy
            if 0 <= bx < ROWS and 0 <= by < COLS and maze[bx][by] == 0 and (bx, by) not in visited:
                visited.add((bx, by))
                queue.appendleft((bx, by, path + [(x, y)]))

    return []  # Jika tidak ditemukan jalur

# Fungsi DFS untuk Patroli
def dfs(maze, start, end):
    stack = [(start[0], start[1], [])]
    visited = set()

    while stack:
        x, y, path = stack.pop()
        if (x, y) == (end[0], end[1]):
            return path + [(x, y)]
        if (x, y) not in visited:
            visited.add((x, y))
            directions = [(-1,0),(1,0),(0,-1),(0,1)]
            random.shuffle(directions)  # Acak arah untuk patroli
            for dx, dy in directions:
                ndx, ndy = x + dx, y + dy
                #print(f"nx: {ndx}, ny: {ndy}, maze shape: {len(maze)}x{len(maze[0])}")
                if 0 <= ndx < len(maze) and 0 <= ndy < len(maze[0]) and
```



```
maze[ndx][ndy] == 0:
                    stack.append((ndx, ndy, path + [(x, y)]))
    return []


def hybrid_ai(maze, enemy_pos, player_pos, game_time):
    global current_algorithm, last_switch

    # Hitung probabilitas BFS berdasarkan waktu (semakin lama semakin
sering BFS)
    bfs_probability = min(0.1 + game_time / 300, 0.9)  # Dari 10% hingga
90%

    # Berganti algoritma setiap 15 detik atau berdasarkan probabilitas
    if time.time() - last_switch > 15:
        if random.random() < bfs_probability:
            current_algorithm = "BFS"
        else:
            current_algorithm = "DFS"
        last_switch = time.time()

    # Pilih algoritma
    if current_algorithm == "BFS":
        path = bfs(maze, enemy_pos, player_pos)
    else:
        path = dfs(maze, enemy_pos, player_pos)

    # Ambil langkah berikutnya
    if path and len(path) > 1:
        return path[1]
    return enemy_pos


# Fungsi untuk menggambar labirin
def draw_maze():
    for row in range(ROWS):
        for col in range(COLS):
            if maze[row][col] == 1:
                pygame.draw.rect(screen, GRAY, (col * GRID_SIZE, row *
```



```
GRID_SIZE, GRID_SIZE, GRID_SIZE))
            else:
                pygame.draw.rect(screen, WHITE, (col * GRID_SIZE, row *
GRID_SIZE, GRID_SIZE, GRID_SIZE), 1)

# Fungsi untuk menggambar player dan musuh
def draw_characters():
    pygame.draw.rect(screen, GREEN, (player_pos[1] * GRID_SIZE,
player_pos[0] * GRID_SIZE, GRID_SIZE, GRID_SIZE))
    pygame.draw.rect(screen, RED, (enemy_pos[1] * GRID_SIZE,
enemy_pos[0] * GRID_SIZE, GRID_SIZE, GRID_SIZE))
    pygame.draw.rect(screen, BLUE, (goal_pos[1] * GRID_SIZE, goal_pos[0]
* GRID_SIZE, GRID_SIZE, GRID_SIZE))

# Game loop
running = True
clock = pygame.time.Clock()
start_time = time.time()
last_enemy_move = 0

while running:
    # current_time = time.time()
    game_time = time.time() - start_time

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # Kontrol player
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP and player_pos[0] > 0 and
maze[player_pos[0]-1][player_pos[1]] == 0:
                player_pos[0] -= 1
            if event.key == pygame.K_DOWN and player_pos[0] < ROWS-1 and
maze[player_pos[0]+1][player_pos[1]] == 0:
                player_pos[0] += 1
            if event.key == pygame.K_LEFT and player_pos[1] > 0 and
maze[player_pos[0]][player_pos[1]-1] == 0:
                player_pos[1] -= 1
```



```
            if event.key == pygame.K_RIGHT and player_pos[1] < COLS-1
and maze[player_pos[0]][player_pos[1]+1] == 0:
                player_pos[1] += 1

    enemy_pos = hybrid_ai(maze, (enemy_pos[0], enemy_pos[1]),
(player_pos[0], player_pos[1]), game_time)



    # Cek jika musuh menangkap player
    if enemy_pos[0] == player_pos[0] and enemy_pos[1] == player_pos[1]:
        print("Game Over! Musuh menangkap Anda!")
        running = False

    # Cek jika player mencapai goal
    if player_pos[0] == goal_pos[0] and player_pos[1] == goal_pos[1]:
        print("Selamat! Anda mencapai tujuan!")
        running = False

    # Render
    screen.fill(WHITE)
    draw_maze()
    draw_characters()
    pygame.display.flip()
    clock.tick(5)

pygame.quit()
sys.exit()
```

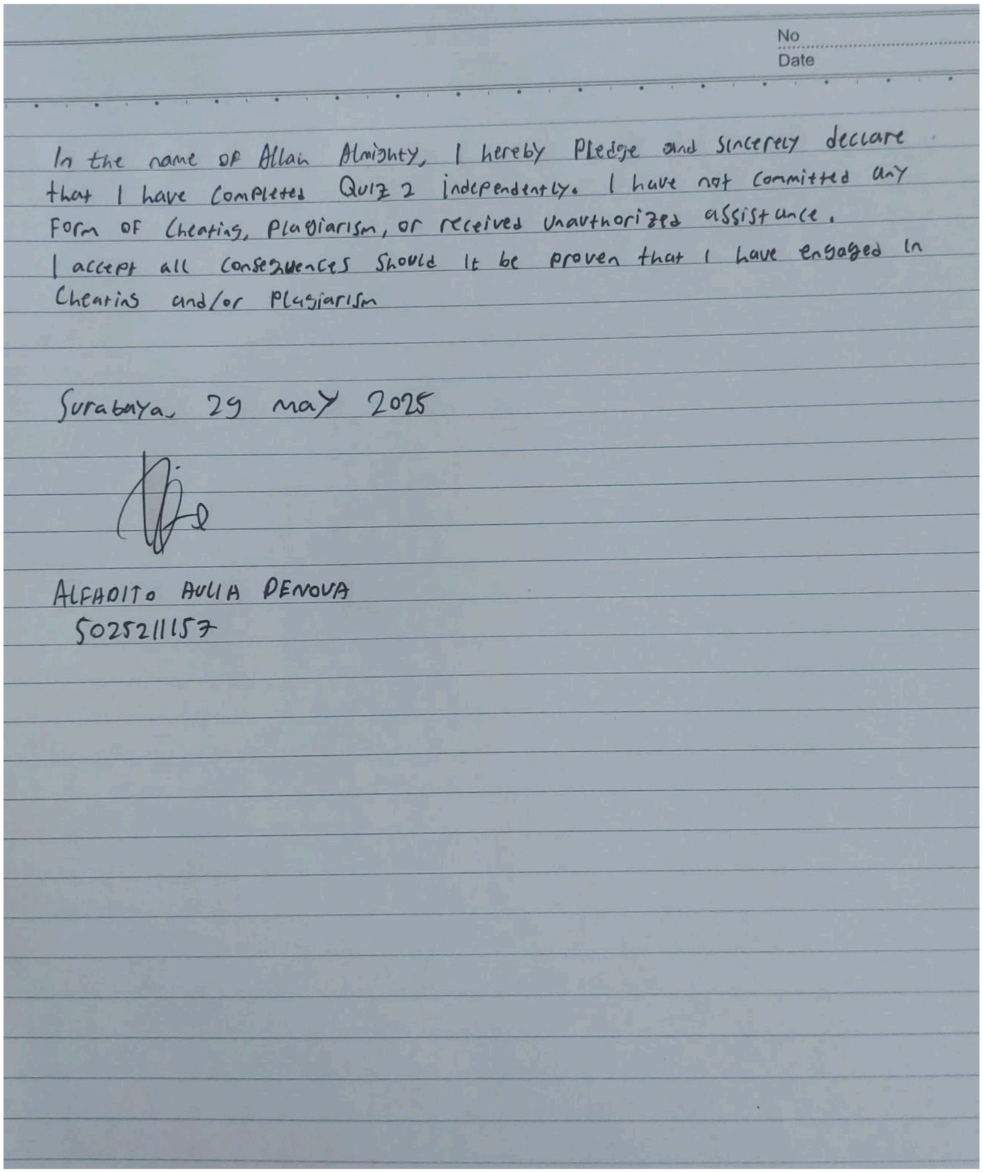
Evaluation of The Game

The game and the algorithm works with a tiny bit of flaw, where the randomness can make the enemy seem more alive, it is also frustrating seeing how the enemy can be a bit dumb. Therefore there could be other search and shortest path algorithms that can be better for this type of game. Like A* or even Dijkstra if the maze is more complex.

Conclusion

For this simple maze game, the BFS and DFS are sufficient, for a bigger and more complex maze, other algorithms are on the table for implementation.

Declaration of Honesty



No

Date

In the name of Allah Almighty, I hereby Pledge and sincerely declare
that I have completed Quiz 2 independently. I have not committed any
form of cheating, plagiarism, or received unauthorized assistance.
I accept all consequences should it be proven that I have engaged in
cheating and/or plagiarism

Surabaya, 29 may 2025

ALFADITO AULIA DENOVA
5025211157

Contribution

Alfadito Aulia Denova - 100%