**PATHFINDING VISUALIZER SOURCE CODE**

import pygame

from queue import PriorityQueue

WIDTH = 1000

WIN = pygame.display.set\_mode((WIDTH, WIDTH))

icon = pygame.image.load('icon.png')

pygame.display.set\_icon(icon)

pygame.display.set\_caption("PathFinder")

RED = (255, 0, 0)

GREEN = (0, 255, 0)

ORANGE = (255, 69, 0)

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

CYAN = (0, 225, 225)

GREY = (128, 128, 128)

class Node:

def \_\_init\_\_(self, row, col, width, total\_rows):

self.row = row

self.col = col

self.x = row \* width

self.y = col \* width

self.colour = WHITE

self.neighbours = []

self.width = width

self.total\_rows = total\_rows

def get\_pos(self):

return self.row, self.col

def is\_open(self):

return self.colour == CYAN

def is\_barrier(self):

return self.colour == BLACK

def is\_start(self):

return self.colour == GREEN

def is\_end(self):

return self.colour == ORANGE

def reset(self):

self.colour = WHITE

def make\_start(self):

self.colour = CYAN

def make\_open(self):

self.colour = GREEN

def make\_barrier(self):

self.colour = BLACK

def make\_end(self):

self.colour = ORANGE

def make\_path(self):

self.colour = RED

def draw(self, win):

pygame.draw.rect(win, self.colour, (self.x, self.y, self.width, self.width))

def update\_neighbours(self, grid):

self.neighbours = []

if self.row < self.total\_rows - 1 and not grid[self.row + 1][self.col].is\_barrier(): # DOWN

self.neighbours.append(grid[self.row + 1][self.col])

if self.row > 0 and not grid[self.row - 1][self.col].is\_barrier(): # UP

self.neighbours.append(grid[self.row - 1][self.col])

if self.col < self.total\_rows - 1 and not grid[self.row][self.col + 1].is\_barrier(): # RIGHT

self.neighbours.append(grid[self.row][self.col + 1])

if self.col > 0 and not grid[self.row][self.col - 1].is\_barrier(): # LEFT

self.neighbours.append(grid[self.row][self.col - 1])

def \_\_lt\_\_(self, other):

return False

def h(p1, p2):

x1, y1 = p1

x2, y2 = p2

return abs(x1 - x2) + abs(y1 - y2)

def reconstruct\_path(came\_from, current, draw):

while current in came\_from:

current = came\_from[current]

current.make\_path()

draw()

def astar(draw, grid, start, end):

count = 0

open\_set = PriorityQueue()

open\_set.put((0, count, start))

came\_from = {}

g\_score = {node: float("inf") for row in grid for node in row}

g\_score[start] = 0

f\_score = {node: float("inf") for row in grid for node in row}

f\_score[start] = h(start.get\_pos(), end.get\_pos())

open\_set\_hash = {start}

while not open\_set.empty():

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

current = open\_set.get()[2]

open\_set\_hash.remove(current)

if current == end:

reconstruct\_path(came\_from, end, draw)

end.make\_end()

return True

for neighbour in current.neighbours:

temp\_g\_score = g\_score[current] + 1

if temp\_g\_score < g\_score[neighbour]:

came\_from[neighbour] = current

g\_score[neighbour] = temp\_g\_score

f\_score[neighbour] = temp\_g\_score + h(neighbour.get\_pos(), end.get\_pos())

if neighbour not in open\_set\_hash:

count += 1

open\_set.put((f\_score[neighbour], count, neighbour))

open\_set\_hash.add(neighbour)

neighbour.make\_open()

draw()

if current != start:

current.make\_open()

return False

def dijkstra(draw, grid, start, end):

count = 0

open\_set = PriorityQueue()

open\_set.put((0, count, start))

came\_from = {}

g\_score = {node: float("inf") for row in grid for node in row}

g\_score[start] = 0

open\_set\_hash = {start}

while not open\_set.empty():

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

current = open\_set.get()[2]

open\_set\_hash.remove(current)

if current == end:

reconstruct\_path(came\_from, end, draw)

end.make\_end()

return True

for neighbour in current.neighbours:

temp\_g\_score = g\_score[current] + 1

if temp\_g\_score < g\_score[neighbour]:

came\_from[neighbour] = current

g\_score[neighbour] = temp\_g\_score

if neighbour not in open\_set\_hash:

count += 1

open\_set.put((g\_score[neighbour], count, neighbour))

open\_set\_hash.add(neighbour)

neighbour.make\_open()

draw()

if current != start:

current.make\_open()

return False

def make\_grid(rows, width):

grid = []

gap = width // rows

for i in range(rows):

grid.append([])

for j in range(rows):

node = Node(i, j, gap, rows)

grid[i].append(node)

return grid

def draw\_grid(win, rows, width):

gap = width // rows

for i in range(rows):

pygame.draw.line(win, GREY, (0, i \* gap), (width, i \* gap))

for j in range(rows):

pygame.draw.line(win, GREY, (j \* gap, 0), (j \* gap, width))

def draw(win, grid, rows, width):

win.fill(WHITE)

for row in grid:

for node in row:

node.draw(win)

draw\_grid(win, rows, width)

pygame.display.update()

def get\_clicked\_pos(pos, rows, width):

gap = width // rows

y, x = pos

row = y // gap

col = x // gap

return row, col

def main(win, width):

algo = int(input("Select the algorithm: 1)A\* 2)Dijkstra\n"))

if pygame.key.get\_pressed():

ROWS = 50

grid = make\_grid(ROWS, width)

start = None

end = None

run = True

while run:

draw(win, grid, ROWS, width)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if pygame.mouse.get\_pressed()[0]: # LEFT

pos = pygame.mouse.get\_pos()

row, col = get\_clicked\_pos(pos, ROWS, width)

node = grid[row][col]

if not start and node != end:

start = node

start.make\_start()

elif not end and node != start:

end = node

end.make\_end()

elif node != end and node != start:

node.make\_barrier()

elif pygame.mouse.get\_pressed()[2]: # RIGHT

pos = pygame.mouse.get\_pos()

row, col = get\_clicked\_pos(pos, ROWS, width)

node = grid[row][col]

node.reset()

if node == start:

start = None

elif node == end:

end = None

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE and start and end:

for row in grid:

for node in row:

node.update\_neighbours(grid)

if algo==2:

dijkstra(lambda: draw(win, grid, ROWS, width), grid, start, end)

else:

astar(lambda: draw(win, grid, ROWS, width), grid, start, end)

if event.key == pygame.K\_c:

start = None

end = None

grid = make\_grid(ROWS, width)

pygame.quit()

main(WIN, WIDTH)