Pr. Mohamed RHAZZAF

Hands-On Exercises 2

Solving the 8-Puzzle Using Uninformed Search (BFS & DFS) in Python

Objectives

- Model the 8-puzzle environment using Python.
- Implement a PuzzleState class with operators (move actions).
- Implement an AgentSolver class that performs BFS and DFS.
- Understand node expansion, frontier, and visited sets.
- Trace and reconstruct solution paths.
- Execute BFS and DFS to solve sample puzzles and compare behavior.

Language: Python (numpy package)

Environment: Vs code or pycharm

Introduction

The 8-puzzle consists of:

- A 3×3 grid with tiles numbered 1–8
- One empty tile represented with 0

Goal state example:

1 2 3
4 5 6
7 8 0

A board in the first place will be presented as [1,2,3,4,5,6,7,8,0]

1- Create the PuzzleState Class

Represent each board configuration as an object with methods to generate successor states.

- The Class PuzzleState Structure

```
class PuzzleState:
    def __init__(self, board, parent=None, move=None, depth=0):
        self.board = board
        self.parent = parent
        self.move = move
        self.depth = depth
```

- Method to find index of the empty tile
- Add this two methods for the Check of equality

```
    def __eq__(self, other):
        return self.board == other.board

    def __hash__(self):
        return hash(self.board)
```

2- Implement the Successor Function

Create method successor to get the list of possible next states:

- Convert 1D Index to 2D Coordinates.
- Define Allowed Moves
- Create New States
    - Copy board
    - Swap positions
    - Create a new PuzzleState instance

3- Design the AgentSolver Class

Handle BFS/DFS and search logic.

- Basic Structure:

```
    class AgentSolver:
        def __init__(self, start_state, goal_state):
            self.start = start_state
            self.goal = goal_state
```

- Create method for checking Goal with given state
- Create method for Solution Reconstruction

4- Implement BFS (Breadth-First Search)

Data Structures

- frontier as queue (using collections.deque)
- visited as a Python set

BFS Algorithm Steps

```
Initialize queue with start state
While queue not empty:
    Pop left element
    If goal → reconstruct path
    Expand node
```

```
        Add successors not in visited

    Return failure if frontier empty
```

5- Implement DFS (Depth-First Search)

Data Structures

- frontier as stack (Python list)
- visited as set

DFS Algorithm Steps

```
    Initialize stack with start state
    While queue not empty:
        Pop element
        If goal → reconstruct path
        Expand node
        Add successors not in visited
    Return failure if frontier empty
```

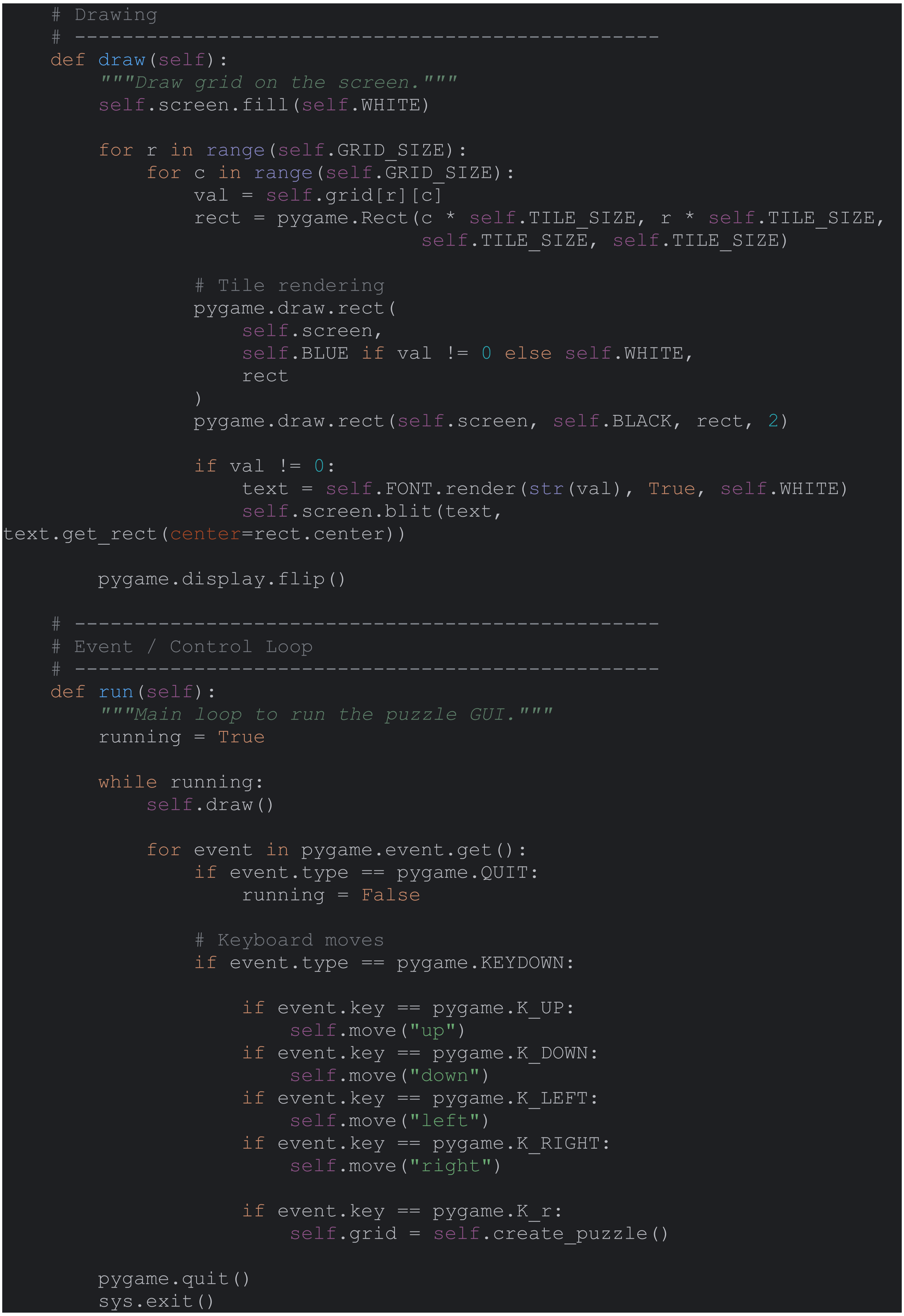
6- Environment Setup and Testing

Create Initial and Goal States, and run BFS and Display Solution

```
from agent_solver import AgentSolver
from puzzle_state import PuzzleState



if __name__ == "__main__":
    start = PuzzleState([1, 4, 2, 3, 0, 5, 6, 7, 8])
    goal = PuzzleState([1, 2, 3, 4, 5, 6, 7, 8, 0])
    solver = AgentSolver(start, goal)
    solution = solver.bfs()
```

modify this class and use it to display the solution in GUI

```
import pygame
import random
import sys
from time import sleep

# ------------------------------------------------------
# 8-PUZZLE GUI CLASS
# ------------------------------------------------------
class EightPuzzleGUI:
    def __init__(self, tile_size=100, grid_size=3):
        pygame.init()
        # --- Settings ---
        self.TILE_SIZE = tile_size
        self.GRID_SIZE = grid_size
        self.WIDTH = tile_size * grid_size
        self.HEIGHT = tile_size * grid_size

        # --- Colors ---
        self.WHITE = (255, 255, 255)
        self.BLACK = (0, 0, 0)
        self.BLUE = (70, 130, 180)

        # --- Font ---
        self.FONT = pygame.font.SysFont("Arial", 40, bold=True)

        # --- Window ---
        self.screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT))
        pygame.display.set_caption("8 Puzzle")

        # --- Puzzle ---
        self.grid = self.create_puzzle()

    # --------------------------------------------------
    # Puzzle Logic
    # --------------------------------------------------
    def create_puzzle(self):
        """Return a shuffled 8-puzzle grid."""
        nums = list(range(1, 9)) + [0]  # 0 = empty tile
        random.shuffle(nums)
        return [nums[i:i + self.GRID_SIZE] for i in range(0, 9,
self.GRID_SIZE)]

    def find_empty(self):
        """Return (row, col) of empty cell."""
        for r in range(self.GRID_SIZE):
            for c in range(self.GRID_SIZE):
                if self.grid[r][c] == 0:
                    return r, c
        return None

    def move(self, direction):
        """Move empty tile in a given direction."""
        r, c = self.find_empty()

        if direction == "up" and r < self.GRID_SIZE - 1:
            self.grid[r][c], self.grid[r + 1][c] = self.grid[r + 1][c],
self.grid[r][c]

        elif direction == "down" and r > 0:
            self.grid[r][c], self.grid[r - 1][c] = self.grid[r - 1][c],
self.grid[r][c]

        elif direction == "left" and c < self.GRID_SIZE - 1:
            self.grid[r][c], self.grid[r][c + 1] = self.grid[r][c + 1],
self.grid[r][c]

        elif direction == "right" and c > 0:
            self.grid[r][c], self.grid[r][c - 1] = self.grid[r][c - 1],
self.grid[r][c]

    # --------------------------------------------------
```



```
    # Drawing
    # -------------------------------------------------
    def draw(self):
        """Draw grid on the screen."""
        self.screen.fill(self.WHITE)

        for r in range(self.GRID_SIZE):
            for c in range(self.GRID_SIZE):
                val = self.grid[r][c]
                rect = pygame.Rect(c * self.TILE_SIZE, r * self.TILE_SIZE,
                                   self.TILE_SIZE, self.TILE_SIZE)

                # Tile rendering
                pygame.draw.rect(
                    self.screen,
                    self.BLUE if val != 0 else self.WHITE,
                    rect
                )
                pygame.draw.rect(self.screen, self.BLACK, rect, 2)

                if val != 0:
                    text = self.FONT.render(str(val), True, self.WHITE)
                    self.screen.blit(text,
text.get_rect(center=rect.center))

        pygame.display.flip()

    # -------------------------------------------------
    # Event / Control Loop
    # -------------------------------------------------
    def run(self):
        """Main loop to run the puzzle GUI."""
        running = True

        while running:
            self.draw()

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

                # Keyboard moves
                if event.type == pygame.KEYDOWN:

                    if event.key == pygame.K_UP:
                        self.move("up")
                    if event.key == pygame.K_DOWN:
                        self.move("down")
                    if event.key == pygame.K_LEFT:
                        self.move("left")
                    if event.key == pygame.K_RIGHT:
                        self.move("right")

                    if event.key == pygame.K_r:
                        self.grid = self.create_puzzle()

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