

## Hands-On Exercises 3

# Implementing Best-First Search, Uniform-Cost Search, and A\* for Grid Navigation

### Objectives

- Represent a grid environment as a 2D list in Python.
- Implement heuristic functions (Euclidean, Manhattan).
- Use priority queues (heapq) to manage frontier nodes.
- Implement search strategies:
  - Best-First Search
  - Uniform-Cost Search
  - A\* Search (with weighted evaluation)
- Reconstruct an action path using parent links.
- Compare different search behaviors in complex environments.

**Language:** Python (numpy package, heapq package)

**Environment:** Vs code or pycharm

### Introduction

The following code is the implementation of our grid world with obstacles for solving the pathfinding problem

The environment is a 30 by 30 grid world with random obstacle positions, the agent has the purpose of finding the best path from a given start position to a destination one

```
import pygame
import random

class PathFindingGUI:
    def __init__(self,
                 grid_width=30,
                 grid_height=20,
                 cell_size=30,
                 obstacle_rate=0.25):
        pygame.init()

        # Configuration
        self.GW = grid_width
        self.GH = grid_height
        self.CELL = cell_size
        self.OBST_RATE = obstacle_rate

        self.WW = self.GW * self.CELL
        self.WH = self.GH * self.CELL

        # Colors
```

```

self.WHITE = (255, 255, 255)
self.BLACK = (0, 0, 0)
self.GRAY = (150, 150, 150)
self.GREEN = (0, 200, 0)      # agent start
self.RED = (200, 50, 50)      # goal

# Window
self.screen = pygame.display.set_mode((self.WW, self.WH))
pygame.display.set_caption("Agent Pathfinding Environment")
self.clock = pygame.time.Clock()

# Environment grid + positions
self.grid = self.generate_obstacles()
self.start_pos = self.random_free_cell()
self.goal_pos = self.random_free_cell()

# -----
# GRID & SPAWNING
# -----
def generate_obstacles(self):
    grid = [[0 for _ in range(self.GW) for _ in range(self.GH)]
    for y in range(self.GH):
        for x in range(self.GW):
            if random.random() < self.OBST_RATE:
                grid[y][x] = 1
    return grid

def random_free_cell(self):
    while True:
        x = random.randint(0, self.GW - 1)
        y = random.randint(0, self.GH - 1)
        if self.grid[y][x] == 0:
            return [x, y]    # mutable for movement

# -----
# DRAWING
# -----
def draw(self):
    self.screen.fill(self.WHITE)

    # Draw grid + obstacles
    for y in range(self.GH):
        for x in range(self.GW):
            rect = pygame.Rect(x * self.CELL, y * self.CELL,
                               self.CELL, self.CELL)

            if self.grid[y][x] == 1:
                pygame.draw.rect(self.screen, self.BLACK, rect)

            pygame.draw.rect(self.screen, self.GRAY, rect, 1)

    # Draw start (agent)
    sx, sy = self.start_pos
    rect_s = pygame.Rect(sx * self.CELL, sy * self.CELL,
                          self.CELL, self.CELL)
    pygame.draw.rect(self.screen, self.GREEN, rect_s)

    # Draw goal
    gx, gy = self.goal_pos
    rect_g = pygame.Rect(gx * self.CELL, gy * self.CELL,
                          self.CELL, self.CELL)

```

```

pygame.draw.rect(self.screen, self.RED, rect_g)

pygame.display.flip()

# -----
# AGENT MOVEMENT
# -----
def handle_keys(self):
    keys = pygame.key.get_pressed()
    new_x, new_y = self.start_pos

    if keys[pygame.K_UP] or keys[pygame.K_w]:
        new_y -= 1
    elif keys[pygame.K_DOWN] or keys[pygame.K_s]:
        new_y += 1
    elif keys[pygame.K_LEFT] or keys[pygame.K_a]:
        new_x -= 1
    elif keys[pygame.K_RIGHT] or keys[pygame.K_d]:
        new_x += 1

    # Validate movement
    if 0 <= new_x < self.GW and 0 <= new_y < self.GH:
        if self.grid[new_y][new_x] == 0:
            self.start_pos = [new_x, new_y]

# -----
# MAIN LOOP
# -----
def run(self):
    running = True
    while running:
        self.clock.tick(10)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        self.handle_keys()
        self.draw()

pygame.quit()

```

## 1- Grid Environment and Agent Solver

You are provided with a skeleton class AgentSolver that contains utility functions and empty algorithm methods.

You must study the provided structure and complete the algorithm implementations.

Grid Format

- 0 → free cell
- 1 → obstacle

Movement allowed: up, down, left, right

```
import heapq
import math

UP_BOLD = "↑"
DOWN_BOLD = "↓"
LEFT_BOLD = "←"
RIGHT_BOLD = "→"

class AgentSolver: 2 usages
    def __init__(self, start_position, goal_position, grid):
        self.start = start_position
        self.goal = goal_position
        self.grid = grid

    def reconstruct_path(self, parent):...

    def heuristic_1(self, state):...

    def heuristic_2(self, state):...

    def best_first(self, heuristic):...

    def ucs(self):...

    def a_star(self, heuristic, coef):...
```

## 2- Provided Class Structure

Students receive the following code base (you do not modify this in the lab description; they will work inside it):

### Key Components to Understand

1. **reconstruct\_path**: Rebuilds the path from goal to start.
2. **heuristic\_1**: Euclidean distance.
3. **heuristic\_2**: Manhattan distance.
4. **best\_first**: Greedy search based only on the heuristic.
5. **ucs**: Uniform-Cost Search using path cost only.
6. **a\_star**: A\* implementation combining cost and heuristic through a weighting factor.

Students will need to analyze how the algorithms explore the grid and produce action directions.

Pr. Mohamed RHAZZAF

### 3- Evaluation of the work

The final code should give the results relative to the following code :

```
from agent_solver import AgentSolver
from path_finding_gui import PathFindingGUI

if __name__ == '__main__':
    env = PathFindingGUI()
    agent = AgentSolver(env.start_pos, env.goal_pos, env.grid)
    path = agent.ucs()
    print("UCS Path:", path, len(path))
    path = agent.best_first(agent.heuristic_1)
    print("Best First search L2 Path:", path, len(path))
    path = agent.best_first(agent.heuristic_2)
    print("Best First search L1 Path:", path, len(path))
    path = agent.a_start(agent.heuristic_2, 0.5)
    print("A* L1 0.5 Path:", path, len(path))
    path = agent.a_start(agent.heuristic_2, 0.1)
    print("A* L1 0.1 Path:", path, len(path))
    path = agent.a_start(agent.heuristic_2, 0.9)
    print("A* L1 0.9 Path:", path, len(path))
    path = agent.a_start(agent.heuristic_1, 0.5)
    print("A* L2 0.5 Path:", path, len(path))
    path = agent.a_start(agent.heuristic_1, 0.1)
    print("A* L2 0.1 Path:", path, len(path))
    path = agent.a_start(agent.heuristic_1, 0.9)
    print("A* L2 0.9 Path:", path, len(path))
    env.run()
```