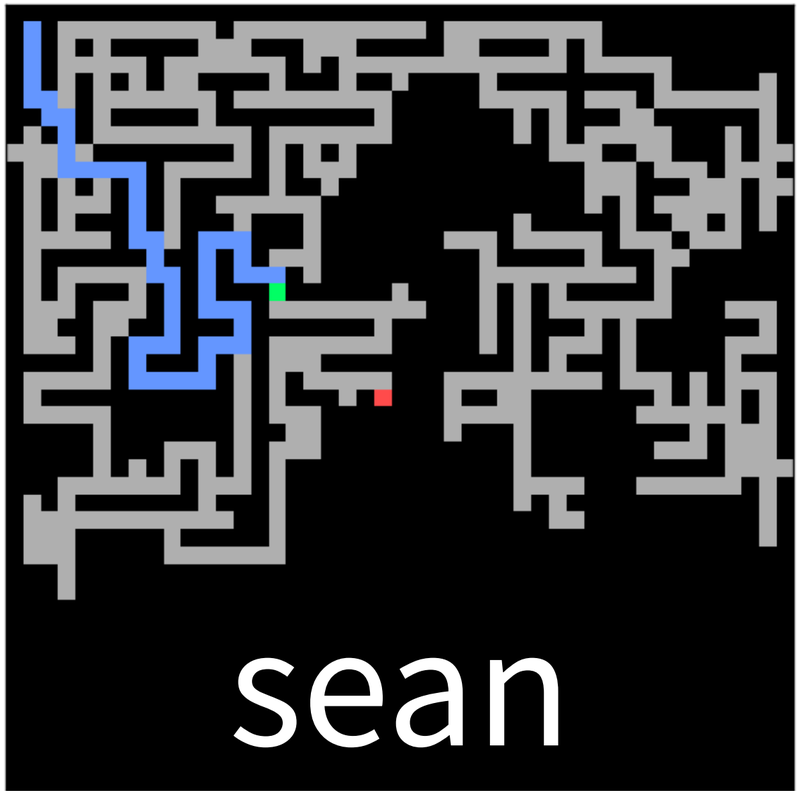
# Maze-RL Demo

```
maze/
├── env-partial/        # Custom Gym-style environments
├── train/       # Exploration and trajectory generation
├── run/         # Run trained policies or visualize agent paths
├── outputs/     # Saved .npy, .jsonl, .json for training
├── visual/      # Rendering utilities
├── requirements.txt
└── .gitignore
```




This project explores reinforcement learning in maze environments with multiple goals and traps. A custom simulator was built using Python and Pygame to test search strategies like BFS, A*, and TSP with theta* smoothing. The agent must efficiently collect all goals while avoiding deadly traps.
Designed for testing multi-goal navigation in partially observable environments

# env_partial:  maze6_multigoals.py

- A custom sizedPOMDP maze with miltiple goal, the agent start at a fixed point to explore the maze find all the goals with limited steps
- using growing tree algorithm with 15% wall removal for path diversity
- real-time rendering using Pygame with color tiles (walls paths goals and agent)

Tools Used:
🐍 Python 3.10
🎮 Gymnasium API
🧱 NumPy for grid processing
🎨 Pygame for visualization

```python
# Step 1: Growing Tree 生成主路徑
start_x, start_y = 1, 1
self.agent_pos = (start_x, start_y)
self.grid[start_x, start_y] = 0
stack = [(start_x, start_y)]

while stack:
    x, y = stack[-1]
    nbs = [n for n in neighbors(x, y) if self.grid[n] == 1]
    if nbs:
        nx, ny = random.choice(nbs)
        self.grid[(x + nx)//2, (y + ny)//2] = 0
        (variable) stack: list[tuple[int, int]]
        stack.append((nx, ny))
    else:
        stack.pop()

# Step 2: 打掉 15% 牆
wall_candidates = [(i, j) for i in range(1, self.size - 1)
                   for j in range(1, self.size - 1) if self.grid[i, j] == 1
random.shuffle(wall_candidates)
for i in range(int(len(wall_candidates) * 0.15)):
    self.grid[wall_candidates[i]] = 0

# Step 3: 隨機挑選 num_goals 個通道格子作為 goal
path_cells = [(i, j) for i in range(self.size)
              for j in range(self.size) if self.grid[i, j] == 0 and (i, j)
self.goal_list = random.sample(path_cells, self.num_goals)

while len(self.goal_list) < self.num_goals:
    gx, gy = random.randint(
        0, self.size-1), random.randint(0, self.size-1)
    if self.grid[gx, gy] == 0 and (gx, gy) not in self.goal_list:
        self.goal_list.append((gx, gy))
```

# Train:train_maze6_multi.py

```python
while success_count < REQUIRED_SUCCESS:
    obs, _ = envs.reset(seed=SEED)
    internal_map = np.full((SIZE, SIZE), -1, dtype=np.int8)
    pos = tuple(obs["position"][0])
    internal_map[pos] = 2
    trajectory = [pos]
    traps_seen = set()

    for step in range(MAX_STEPS):
        action = envs.single_action_space.sample()
        next_obs, reward, terminated, truncated, info = envs.step([action])
        next_pos = tuple(next_obs["position"][0])

        if pos == next_pos:
            dx, dy = maze6.MOVE[action]
            wall_pos = (pos[0] + dx, pos[1] + dy)
            if 0 <= wall_pos[0] < SIZE and 0 <= wall_pos[1] < SIZE:
                internal_map[wall_pos] = 0
        else:
            internal_map[next_pos] = 2
            pos = next_pos
            trajectory.append(next_pos)

        if pos in full_env.traps:
            traps_seen.add(pos)
```

- This logic dynamically constructs an internal map of the environment.
- Traps are detected on contact and logged separately.
- Trajectories are saved for each episode.

```
PS C:\Users\seana\maze\train> python .\train_maze6_multi.py
🗺️ 地圖儲存於：C:/Users/seana/maze/outputs/mem_trap/gt_maze6_multi_101x101_SEED15.npy
🎯 成功收集全部 goals！（目前成功次數：1）
🎯 成功收集全部 goals！（目前成功次數：2）
💾 已儲存探索紀錄，共 14 筆：C:/Users/seana/maze/outputs/mem_trap/maze6_multi_4demo.jsonl
```

```python
results.append({
    "explored_map": (internal_map == 2).astype(np.uint8).tolist(),
    "known_walls": (internal_map == 0).astype(np.uint8).tolist(),
    "known_traps": [list(map(int, t)) for t in traps_seen],
    "start_pos": list(map(int, full_env.agent_pos)),
    "goal_list": [list(map(int, g)) for g in full_env.goal_list],
    "collected_goals": [list(map(int, g)) for g in full_env.collected_goals],
    "trajectory": [[int(x), int(y)] for x, y in trajectory],
    "maze_id": f"maze6_multi_ep{attempts+1}"
})
```


```
{
 "explored_map": [[0, 0], [1, 0], [2, 0], ..., [36, 27]],
 "start_pos": [0, 0],
 "goal_list": [[36, 27], [35, 17], [17, 29]],
 "collected_goals": [[36, 27]],
 "trajectory": [[0, 0], [1, 0], [2, 0], ..., [36, 27]],
 "known_walls": [[0, 1], [1, 1], ..., [35, 28]],
  "maze_id": "gt_maze6_multi_100x100_SEED1061"
}
```

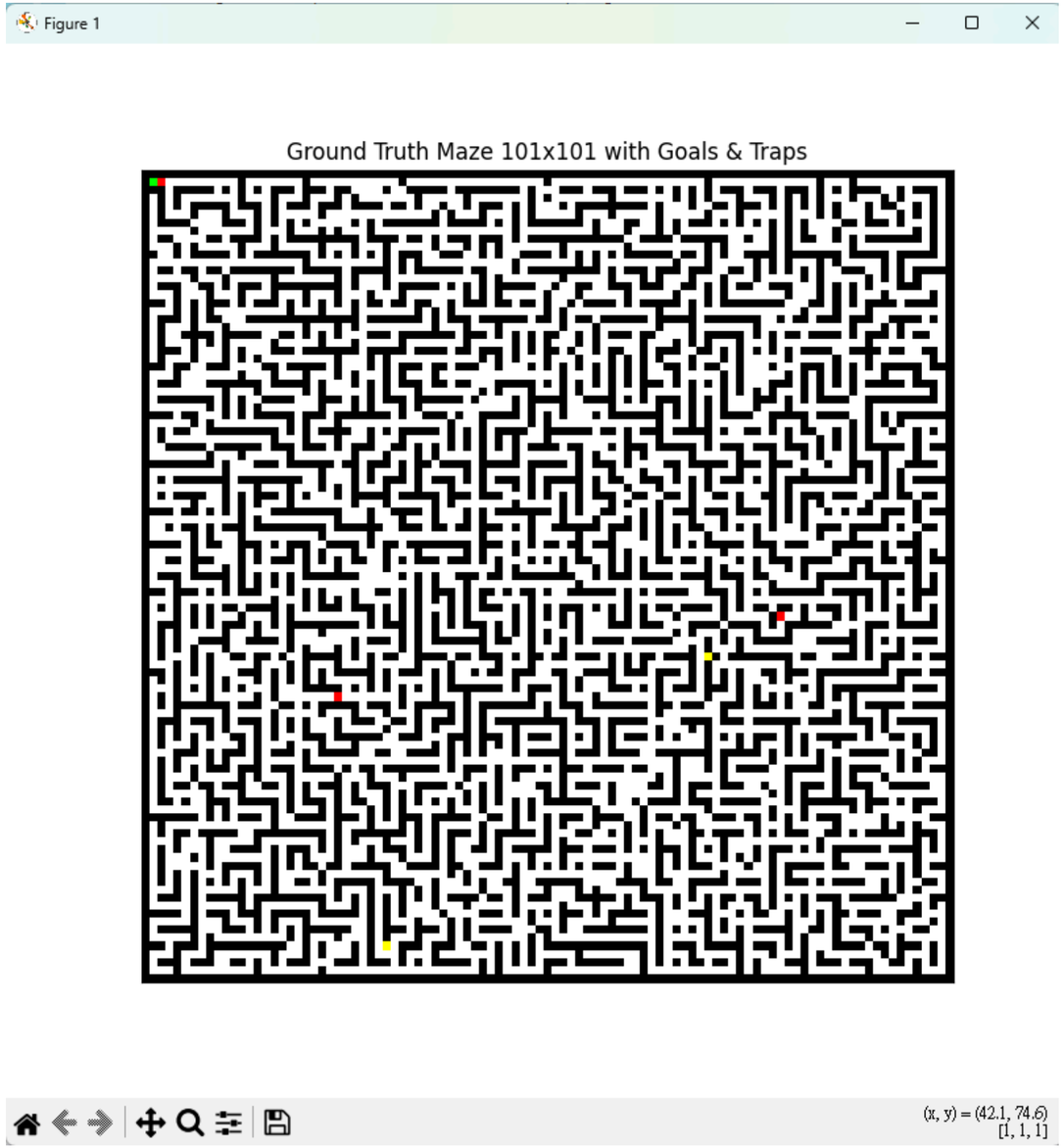

This snippet appends structured data of the agent’ s exploration into a list of dictionaries (results). The log includes the visible map, identified walls, traps, goal positions, and the full path (trajectory). This format is ideal for use in reinforcement learning analysis and language model training.

```python
gt_path = f"C:/Users/seana/maze/outputs/mem_trap/gt_maze6_multi_{SIZE}x{SIZE}_SEED{SEED}.npy"
os.makedirs(os.path.dirname(gt_path), exist_ok=True)
np.save(gt_path, gt)
print(f"\U0001F5FA\ufe0f 地圖儲存於：{gt_path}")
```

- generates a ground truth map

# Visual:v3.py(for multi goals

```python
import numpy as np
import matplotlib.pyplot as plt

# 讀取 ground truth
gt_path = "C:/Users/seana/maze/outputs/mem_trap/gt_maze6_multi_100x100_SEED1525.npy"
data = np.load(gt_path, allow_pickle=True).item()

maze = data["wall_map"]
start = data["start_pos"]
goals = data["goal_list"]
traps = set(map(tuple, data["trap_list"]))

# 繪圖
H, W = maze.shape
img = np.zeros((H, W, 3))
for i in range(H):
    for j in range(W):
        if maze[i, j] == 1:
            img[i, j] = (0.0, 0.0, 0.0)     # 牆
        else:
            img[i, j] = (1.0, 1.0, 1.0)     # 路

for tx, ty in traps:
    img[tx, ty] = (1.0, 1.0, 0.0)  # 陷阱：黃

for gx, gy in goals:
    img[gx, gy] = (1.0, 0.0, 0.0)  # 目標：紅

img[start[0], start[1]] = (0.0, 1.0, 0.0)   # 起點：綠

plt.figure(figsize=(8, 8))
plt.imshow(img, interpolation='nearest')
plt.title(f"Ground Truth Maze {H}x{W} with Goals & Traps")
plt.xticks([]), plt.yticks([])
plt.show()
```




traps:

goals: start_pos:

# Run:run_tsp_theta_6

```
def theta_star(source, target):
    heap = [(0, 0, source, source, [source])]
    visited = set()
    while heap:
        f, g, cur, parent, path = heapq.heappop(heap)
        if cur in visited:
            continue
        visited.add(cur)
        if cur == target:
            return path
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                if dx == 0 and dy == 0:
                    continue
                nx, ny = cur[0] + dx, cur[1] + dy
                if 0 <= nx < H and 0 <= ny < W and combined[nx, ny] == 0 and (nx, ny) not in trap_set:
                    next_pos = (nx, ny)
                    if line_of_sight(parent, next_pos):
                        heapq.heappush(heap, (g + np.hypot(*(np.subtract(next_pos, parent))) + np.hypot(*(np.subtract(
                            target, next_pos))), g + np.hypot(*(np.subtract(next_pos, parent))), next_pos, parent, path + [next_pos]))
                    else:
                        heapq.heappush(heap, (g + np.hypot(*(np.subtract(next_pos, cur))) + np.hypot(*(np.subtract(
                            target, next_pos))), g + np.hypot(*(np.subtract(next_pos, cur))), next_pos, cur, path + [next_pos]))
    return []
```

- “Theta* enhances A* by enabling diagonal shortcuts and real-world movement behavior — fast, flexible, and trap-aware.”

# Multi goals :TSP

This block solves a maze-based version of the Traveling Salesman Problem (TSP):

- It precomputes all pairwise shortest paths between the starting point and goals using the Theta* algorithm.
- Then, it tries every permutation (order) of the goals.
- For each permutation, it concatenates the paths in order and checks:
  - If all required segments are valid.
  - If the total path length is shorter than previous solutions.
- The final result is the shortest possible route visiting all goals, avoiding walls and traps.

```
# === 建立所有合法 pair 間的路徑 ===
all_points = [start] + goals
paths = {}
for a, b in itertools.permutations(all_points, 2):
    path = theta_star(a, b)
    if path:
        paths[(a, b)] = path

# === 嘗試所有 goal 排列，選擇避開陷阱的最短組合 ===
best_path = None
min_length = float("inf")
for perm in itertools.permutations(goals):
    full = []
    current = start
    valid = True
    for g in perm:
        key = (current, g)
        if key not in paths:
            valid = False
            break
        segment = paths[key]
        if full:
            segment = segment[1:]
        full.extend(segment)
        current = g
    if valid and len(full) < min_length:
        min_length = len(full)
        best_path = full

print("\U0001F9ED 最短合法路徑長度（避開陷阱）：", min_length)
```

## ■ Demo: TSP Route Planning in Maze

This video showcases an agent navigating a partially observable maze with multiple goals and traps using a TSP strategy powered by Theta* algorithm. The agent dynamically finds the shortest trap-free path through all goals.

■ Watch Demo: [run_demo.mp4] (Attached separately)