实验一：环境配置及八数码问题

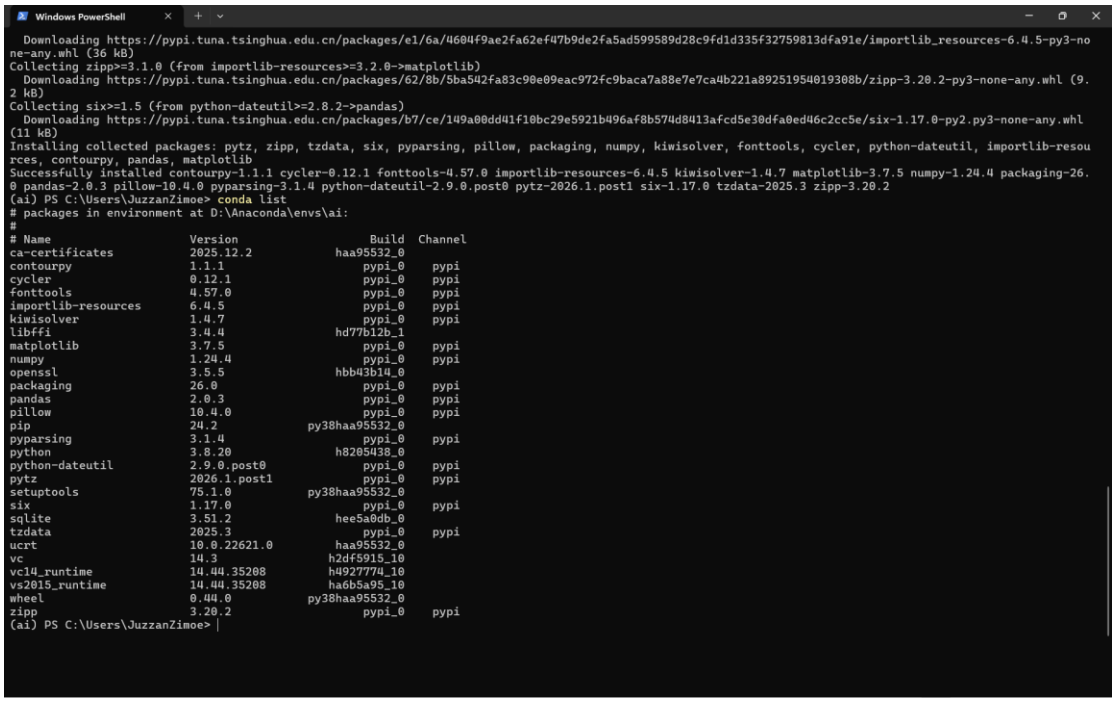
一、环境配置与安装

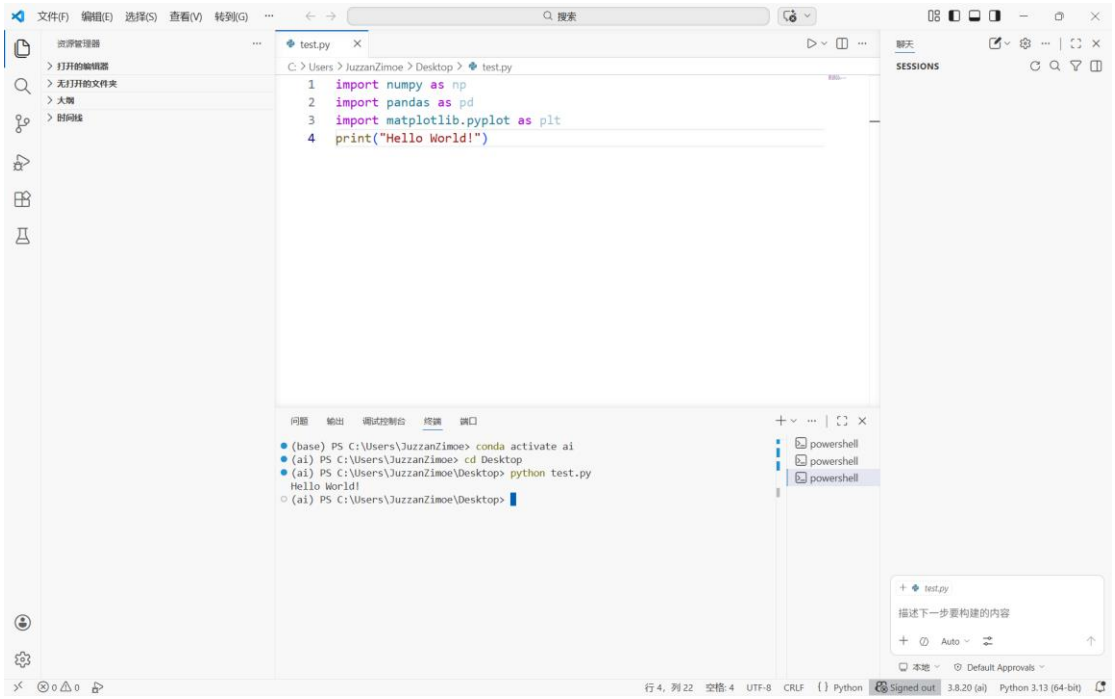
安装 Vscode 和常用插件

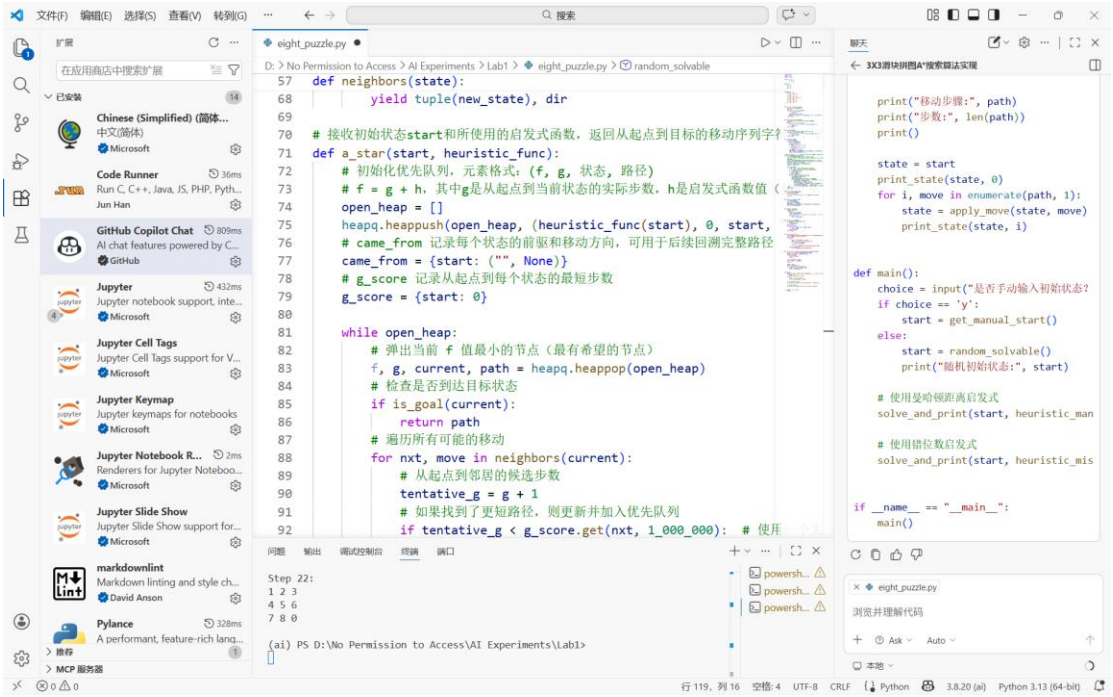
安装 Anaconda 并创建虚拟环境，熟悉常用的环境管理指令

安装并学习掌握常用 python 库如 pandas、numpy、matplotlib 的使用

提交安装完成截图







二、八数码问题：

通过 AI 辅助编程实一个八数码问题的求解。棋盘为 3×3 的方格，包含数 字 0–8，其中 0 表示空格白板，数字可以通过与空格交换在棋盘上向上下左右移动。给定一个随机的初始状态，程序需要通过 A*搜索方法寻找一系列合法移动，使滑块 1、2、3、4、6、7、8 移动到其正确位置；其中数字 5 的位置不作要求。当这些指定数字均位于目标位置时，即认为达到目标状态。程序需要输出求解过程以及最终的移动步数。

（1）调试并跑通 AI 生成的代码，并对其中 A*搜索部分的代码进行理解与注释。

（2）设计两种或以上的启发式函数。给出每种函数的算法中从一个随机或设定的初始状态开始，给出求解路径、每一步棋盘状态及总移动步数。

（一）程序源码（包括对 A*搜索部分代码的注释）：

```
import heapq
import sys
import random
GOAL = (1, 2, 3, 4, 5, 6, 7, 8, 0)
MOVES = {"U": -3,"D": 3,"L": -1,"R": 1,}
NEIGHBORS = {
    0: ("D", "R"),
    1: ("D", "L", "R"),
    2: ("D", "L"),
    3: ("U", "D", "R"),
    4: ("U", "D", "L", "R"),
    5: ("U", "D", "L"),
    6: ("U", "R"),
    7: ("U", "L", "R"),
    8: ("U", "L"),
```

```
}

def heuristic_manhattan(state):
    # 曼哈顿距离：忽略数字 5 和 0
    dist = 0
    for idx, v in enumerate(state):
        if v == 0 or v == 5:
            continue
        goal_idx = GOAL.index(v)
        dist += abs((idx//3) - (goal_idx//3)) + abs((idx%3) - (goal_idx%3))
    return dist

def heuristic_misplaced(state):
    # 错位数：计算 1/2/3/4/6/7/8 中不在正确位置的数量
    count = 0
    for idx, v in enumerate(state):
        if v in (1, 2, 3, 4, 6, 7, 8) and GOAL[idx] != v:
            count += 1
    return count

def is_goal(state):
    # 只检查 1/2/3/4/6/7/8 是否处于 GOAL 对应位置
    for idx, v in enumerate(state):
        if v in (1, 2, 3, 4, 6, 7, 8) and GOAL[idx] != v:
            return False
    return True

def neighbors(state):
    z = state.index(0)
    for dir in NEIGHBORS[z]:
        ni = z + MOVES[dir]
        # 水平边界检查
        if dir == "L" and z % 3 == 0:
            continue
        if dir == "R" and z % 3 == 2:
            continue
        new_state = list(state)
        new_state[z], new_state[ni] = new_state[ni], new_state[z]
        yield tuple(new_state), dir

# 接收初始状态 start 和启发式函数，返回从起点到目标的移动序列字符串，若无解返回 None
def a_star(start, heuristic_func):
    # 初始化优先队列，元素格式：(f, g, 状态, 路径)
    # f = g + h，其中 g 是从起点到当前状态的实际步数，h 是启发式函数值
    open_heap = []
    heapq.heappush(open_heap, (heuristic_func(start), 0, start, ""))
    # (f, g, state, path)
    # came_from 记录每个状态的前驱和移动方向，可用于后续回溯完整路径
    came_from = {start: ("", None)}
    # g_score 记录从起点到每个状态的最短步数
    g_score = {start: 0}

    while open_heap:
        # 弹出当前 f 值最小的节点（最有希望的节点）
        f, g, current, path = heapq.heappop(open_heap)
        # 检查是否到达目标状态
        if is_goal(current):
            return path
```

```
        # 遍历所有可能的移动
        for nxt, move in neighbors(current):
            # 从起点到邻居的候选步数
            tentative_g = g + 1
            # 如果找到了更短路径，则更新并加入优先队列
            if tentative_g < g_score.get(nxt, 1_000_000):
            # 使用一个大数表示无穷大
                g_score[nxt] = tentative_g
                came_from[nxt] = (move, current)
                # 启发式函数估计当前状态到目标的距离，其可采纳性保证了 A 的最优性
                h = heuristic_func(nxt)     # 计算启发式值
                heapq.heappush(open_heap, (tentative_g + h, tentative_g, nxt,
path + move))
    # 无解
    return None

def print_state(state, step):
    print(f"Step {step}:")
    for i in range(3):
        print(" ".join(str(x) for x in state[i*3:(i+1)*3]))
    print()

def apply_move(state, move):
    z = state.index(0)
    ni = z + MOVES[move]
    new_state = list(state)
    new_state[z], new_state[ni] = new_state[ni], new_state[z]
    return tuple(new_state)

def random_solvable():
    # 生成一个可解状态
    while True:
        perm = list(range(9))
        random.shuffle(perm)
        inv = sum(1 for i in range(9) for j in range(i + 1, 9) if perm[i] and
perm[j] and perm[i] > perm[j])
        if inv % 2 == 0:
            return tuple(perm)

def is_valid_state(state):
    # 检查是否为 9 个数字，0-8 各一次
    if len(state) != 9:
        return False
    nums = set()
    for num in state:
        if not (0 <= num <= 8) or num in nums:
            return False
        nums.add(num)
    return True

def get_manual_start():
    print("请输入初始状态：9 个数字（0-8），用空格分隔，表示 3x3 棋盘的状态（0 表示空
格）。")
    print("例如：1 2 3 4 5 6 7 8 0")
    while True:
        try:
            inp = input("输入：").strip()
            nums = [int(x) for x in inp.split()]
```

```
            if is_valid_state(nums):
                return tuple(nums)
            else:
                print("输入无效：请确保 9 个数字，0-8 各一次。")
        except ValueError:
            print("输入无效：请确保输入数字。")

def solve_and_print(start, heuristic_func, heuristic_name):
    print(f"\n 使用启发式函数：{heuristic_name}")
    path = a_star(start, heuristic_func)
    if path is None:
        print("未找到解")
        return

    print("移动步骤:", path)
    print("步数:", len(path))
    print()

    state = start
    print_state(state, 0)
    for i, move in enumerate(path, 1):
        state = apply_move(state, move)
        print_state(state, i)

def main():
    choice = input("是否手动输入初始状态？(y/n): ").strip().lower()
    if choice == 'y':
        start = get_manual_start()
    else:
        start = random_solvable()
        print("随机初始状态:", start)

    # 使用曼哈顿距离启发式
    solve_and_print(start, heuristic_manhattan, "曼哈顿距离")

    # 使用错位数启发式
    solve_and_print(start, heuristic_misplaced, "错位数")

if __name__ == "__main__":
    main()
```

（二）与 AI 交互的过程（包括提示词等）：

第一阶段：生成基础代码框架

提示词：请你编写代码：棋盘为 3×3 的方格，包含数字 0–8，其中 0 表示空格（白板），数字可以通过与空格交换在棋盘上向上、下、左、右移动。给定一个随机的初始状态，程序需要通过 A*搜索方法寻找一系列合法移动，使滑块 1、2、3、4、6、7、8 移动到其正确位置；其中数字 5 的位置不作要求。当这些指定数字均位于目标位置时，即认为达到目标状态。程序需要输出求解过程以及最终的移动步数。

第二阶段：优化代码，期望将八数码问题的解决过程可视化呈现出来

提示词：帮我修改上面的代码，让它能输出每一步的棋盘状态（用 3×3 网格输出，数字用空格隔开，保持美观）。代码已能得到移动路径，请在得到路径后，从初始状态开始，一步步应用每个移动，每走一步就打印当前棋盘，并标明当前步数，初始状态也要打印（第 0 步）。保持其他功能不变。

第三阶段：优化代码，希望能够实现随机生成或手动输入初始状态

提示词：请添加一个功能，即在移动开始前，让我自己选择：初始状态由程序随机生成，还是由我手动输入初始状态。如果我选择人工输入初始状态的话，你最好提示我输入的格式。其他功能保持不变。

第四阶段：实现使用两种启发式函数求解

提示词：修改之前的代码，之前的代码中目前只用了曼哈顿距离作为启发式，请设计两种启发式函数，给出每种启发式函数的算法从一个随机或设定的初始状态开始到目标状态的步骤，给出求解路径、每一步棋盘状态及总移动步数。其他功能保持不变。

由此我们得到了最终的代码。

（三）启发式函数的定义及说明：

1. 启发式函数的作用

在 A*搜索中，启发式函数$h(n)$用于估计从当前状态n到目标状态的代价（此处为最少移动步数）。它引导搜索优先扩展“有希望”的节点，从而加速找到最优解。启发式函数的可采纳性（即 $h(n)$ 不大于实际最小代价）保证了 A*算法的最优性。

2. 代码中使用的两种启发式函数

①曼哈顿距离（Manhattan Distance）

曼哈顿距离是指在网格中，两点之间只能沿水平或垂直方向移动时的最短距离。对于八数码问题，我们计算每个需要归位的数字（1、2、3、4、6、7、8）从当前位置到其目标位置的曼哈顿距离之和，并忽略空格 0 和数字 5。

计算公式$h_{\text{manhattan}}(state) = \sum_{v\in\{1,2,3,4,6,7,8\}} \left(| r_v^{\text{cur}} - r_v^{\text{goal}} | + | c_v^{\text{cur}} - c_v^{\text{goal}} | \right)$

其中$(r_v^{\text{cur}}, c_v^{\text{cur}})$ 是数字 v 在当前棋盘中的行、列坐标（行和列从 0 开始）。

$\left(r_v^{\text{goal}}, c_v^{\text{goal}}\right)$ 是数字 v 在目标棋盘中的坐标（根据 GOAL 元组确定）。

②错位数（Misplaced Tiles）

错位数启发式统计当前状态中，需要归位的数字（1、2、3、4、6、7、8）有多少个不在其正确位置上。该启发式是曼哈顿距离的一个简化版本，计算更简单，但估计精度较低。

计算公式$h_{\text{misplaced}}(state) = \sum_{v\in\{1,2,3,4,6,7,8\}} \mathbb{1}_{\{state(r_v^{\text{cur}},c_v^{\text{cur}})\neq v\}}$

其中$\mathbb{1}$是指示函数，当数字 v 不在其目标位置时计 1，否则计 0。

（四）代码运行结果：

1. 随机生成初始状态的情况下：

```
(ai) PS D:\No Permission to Access\AI Experiments\Lab1> python eight_puzzle.py
是否手动输入初始状态? (y/n): n
随机初始状态: (6, 3, 1, 8, 4, 0, 2, 5, 7)

使用启发式函数: 曼哈顿距离
移动步骤: ULDRDLLUURDLDRURDLULDRULURDRD
步数: 29

Step 0:
6 3 1
8 4 0
2 5 7

Step 1:
6 3 0
8 4 1
2 5 7

Step 2:
6 0 3
8 4 1
2 5 7

Step 3:
6 4 3
8 0 1
2 5 7

Step 4:
6 4 3
8 1 0
2 5 7

Step 5:
6 4 3
8 1 7
2 5 0

Step 6:
6 4 3
```

```
8 1 7
2 0 5

Step 7:
6 4 3
8 1 7
0 2 5

Step 8:
6 4 3
0 1 7
8 2 5

Step 9:
0 4 3
6 1 7
8 2 5

Step 10:
4 0 3
6 1 7
8 2 5

Step 11:
4 1 3
6 0 7
8 2 5

Step 12:
4 1 3
0 6 7
8 2 5

Step 13:
4 1 3
8 6 7
0 2 5

Step 14:
4 1 3
8 6 7
2 0 5

Step 15:
4 1 3
8 0 7
2 6 5

Step 16:
4 1 3
8 7 0
2 6 5

Step 17:
4 1 3
8 7 5
2 6 0
```

```
Step 18:
4 1 3
8 7 5
2 0 6

Step 19:
4 1 3
8 0 5
2 7 6

Step 20:
4 1 3
0 8 5
2 7 6

Step 21:
4 1 3
2 8 5
0 7 6

Step 22:
4 1 3
2 8 5
7 0 6

Step 23:
4 1 3
2 0 5
7 8 6

Step 24:
4 1 3
0 2 5
7 8 6

Step 25:
0 1 3
4 2 5
7 8 6

Step 26:
1 0 3
4 2 5
7 8 6

Step 27:
1 2 3
4 0 5
7 8 6

Step 28:
1 2 3
4 5 0
7 8 6

Step 29:
1 2 3
4 5 6
```

```
7 8 0


使用启发式函数：错位数
移动步骤：ULDRDLLUURDLDRURDLULDRULURDRD
步数：29

Step 0:
6 3 1
8 4 0
2 5 7

Step 1:
6 3 0
8 4 1
2 5 7

Step 2:
6 0 3
8 4 1
2 5 7

Step 3:
6 4 3
8 0 1
2 5 7

Step 4:
6 4 3
8 1 0
2 5 7

Step 5:
6 4 3
8 1 7
2 5 0

Step 6:
6 4 3
8 1 7
2 0 5

Step 7:
6 4 3
8 1 7
0 2 5

Step 8:
6 4 3
0 1 7
8 2 5

Step 9:
0 4 3
6 1 7
8 2 5

Step 10:
```

```
4 0 3
6 1 7
8 2 5

Step 11:
4 1 3
6 0 7
8 2 5

Step 12:
4 1 3
0 6 7
8 2 5

Step 13:
4 1 3
8 6 7
0 2 5

Step 14:
4 1 3
8 6 7
2 0 5

Step 15:
4 1 3
8 0 7
2 6 5

Step 16:
4 1 3
8 7 0
2 6 5

Step 17:
4 1 3
8 7 5
2 6 0

Step 18:
4 1 3
8 7 5
2 0 6

Step 19:
4 1 3
8 0 5
2 7 6

Step 20:
4 1 3
0 8 5
2 7 6

Step 21:
4 1 3
2 8 5
0 7 6
```

```
Step 22:
4 1 3
2 8 5
7 0 6

Step 23:
4 1 3
2 0 5
7 8 6

Step 24:
4 1 3
0 2 5
7 8 6

Step 25:
0 1 3
4 2 5
7 8 6

Step 26:
1 0 3
4 2 5
7 8 6

Step 27:
1 2 3
4 0 5
7 8 6

Step 28:
1 2 3
4 5 0
7 8 6

Step 29:
1 2 3
4 5 6
7 8 0
```

2.手动设定初始状态的情况下：

```
(ai) PS D:\No Permission to Access\AI Experiments\Lab1> python eight_puzzle.py
是否手动输入初始状态？(y/n): y
请输入初始状态：9 个数字（0-8），用空格分隔，表示 3x3 棋盘的状态（0 表示空格）。
例如：1 2 3 4 5 6 7 8 0
输入：4 6 7 0 5 8 1 2 3

使用启发式函数：曼哈顿距离
移动步骤: DRURDLURULDRDLULDRRUULDLURD
步数: 27

Step 0:
4 6 7
0 5 8
1 2 3
```

```
Step 1:
4 6 7
1 5 8
0 2 3

Step 2:
4 6 7
1 5 8
2 0 3

Step 3:
4 6 7
1 0 8
2 5 3

Step 4:
4 6 7
1 8 0
2 5 3

Step 5:
4 6 7
1 8 3
2 5 0

Step 6:
4 6 7
1 8 3
2 0 5

Step 7:
4 6 7
1 0 3
2 8 5

Step 8:
4 6 7
1 3 0
2 8 5

Step 9:
4 6 0
1 3 7
2 8 5

Step 10:
4 0 6
1 3 7
2 8 5

Step 11:
4 3 6
1 0 7
2 8 5

Step 12:
4 3 6
```

```
1 7 0
2 8 5

Step 13:
4 3 6
1 7 5
2 8 0

Step 14:
4 3 6
1 7 5
2 0 8

Step 15:
4 3 6
1 0 5
2 7 8

Step 16:
4 3 6
0 1 5
2 7 8

Step 17:
4 3 6
2 1 5
0 7 8

Step 18:
4 3 6
2 1 5
7 0 8

Step 19:
4 3 6
2 1 5
7 8 0

Step 20:
4 3 6
2 1 0
7 8 5

Step 21:
4 3 0
2 1 6
7 8 5

Step 22:
4 0 3
2 1 6
7 8 5

Step 23:
4 1 3
2 0 6
7 8 5
```

```
Step 24:
4 1 3
0 2 6
7 8 5

Step 25:
0 1 3
4 2 6
7 8 5

Step 26:
1 0 3
4 2 6
7 8 5

Step 27:
1 2 3
4 0 6
7 8 5


使用启发式函数：错位数
移动步骤：DRURDLURULDRDLULDRRUULDLURD
步数：27

Step 0:
4 6 7
0 5 8
1 2 3

Step 1:
4 6 7
1 5 8
0 2 3

Step 2:
4 6 7
1 5 8
2 0 3

Step 3:
4 6 7
1 0 8
2 5 3

Step 4:
4 6 7
1 8 0
2 5 3

Step 5:
4 6 7
1 8 3
2 5 0

Step 6:
4 6 7
1 8 3
```

```
2 0 5

Step 7:
4 6 7
1 0 3
2 8 5

Step 8:
4 6 7
1 3 0
2 8 5

Step 9:
4 6 0
1 3 7
2 8 5

Step 10:
4 0 6
1 3 7
2 8 5

Step 11:
4 3 6
1 0 7
2 8 5

Step 12:
4 3 6
1 7 0
2 8 5

Step 13:
4 3 6
1 7 5
2 8 0

Step 14:
4 3 6
1 7 5
2 0 8

Step 15:
4 3 6
1 0 5
2 7 8

Step 16:
4 3 6
0 1 5
2 7 8

Step 17:
4 3 6
2 1 5
0 7 8

Step 18:
```

```
4 3 6
2 1 5
7 0 8

Step 19:
4 3 6
2 1 5
7 8 0

Step 20:
4 3 6
2 1 0
7 8 5

Step 21:
4 3 0
2 1 6
7 8 5

Step 22:
4 0 3
2 1 6
7 8 5

Step 23:
4 1 3
2 0 6
7 8 5

Step 24:
4 1 3
0 2 6
7 8 5

Step 25:
0 1 3
4 2 6
7 8 5

Step 26:
1 0 3
4 2 6
7 8 5

Step 27:
1 2 3
4 0 6
7 8 5
```