【期末大作业】智慧港口自动化调度系统 (Smart Port Scheduler)

1. 项目概述

背景：

在一个现代化全自动港口中，你需要编写一个中央控制算法，通过调度机器人（Robot）和轮船（Ship）协同工作，将源源不断产生的货物（Goods）从仓库区搬运至泊位（Berth），并通过轮船运输变现。

目标：

在规定的 1000 帧（Frames） 时间限制内，通过合理的路径规划和任务分配，尽可能多地赚取资金。

2. 地图与坐标系

- **地图大小：** 100×100 的网格（$N = 100$）。
- **坐标系：**
  - **X轴 (行)：** 垂直向下，范围 $[0, 99]$。
  - **Y轴 (列)：** 水平向右，范围 $[0, 99]$。
  - 坐标表示为 `(x, y)`，对应二维数组 `map[x][y]`。
- **地图字符定义：**
  - `.` **空地 (Land)：** 机器人可自由通行。
  - `*` **海洋 (Sea)：** 机器人不可通行，不可停留。
  - `#` **障碍 (Obstacle)：** 机器人不可通行，不可停留。
  - `A` **机器人起始点 (Start)：** 游戏开始时机器人生成的位置（视为 `.`，之后无特殊含义）。
  - `B` **泊位 (Berth)：** 机器人可在此处卸货。**注意：泊位也是可通行的平地**，机器人可以在上面移动。

3. 游戏进程与交互机制

系统采用**标准输入输出 (Standard I/O)** 进行回合制交互。程序启动后，将不断循环执行以下步骤，直到游戏结束。

3.1 帧循环 (Game Loop)

每一帧（Frame）按以下顺序严格执行：

1. **环境更新：** 判题器生成新货物，移除过期货物。
2. **发送状态：** 判题器通过 `stdin` 向你的程序发送当前帧的完整数据（地图快照、机器人状态、货物列表）。
3. **程序决策：** 你的程序读取数据，计算策略。
4. **发送指令：** 你的程序通过 `stdout` 发送控制指令（移动、取货等），以 `OK` 结尾。
5. **判题结算：** 判题器按顺序执行指令，处理移动冲突，计算得分。

3.2 结算顺序 (重要)

在判题器内部，一帧内的动作按以下优先级结算：

1. **移动 (Move):** 所有机器人尝试移动。如果发生碰撞，则停在原地。
2. **交互 (Get/Pull):** 结算取货和卸货操作。
3. **运输 (Go):** 结算轮船的运输卖货操作。

---

4. 实体与行为定义

4.1 机器人 (Robot)

- **数量：** 固定 10 个（ID: 0-9）。
- **属性：**
  - `x, y`: 当前坐标。
  - `has_goods`: 携带货物状态（0:未携带, 1:携带）。
  - `status`: 状态（1:正常, 0:恢复中/异常，本简化版中常驻为1）。
- **指令限制：** 每个机器人每帧只能执行 **1条** 指令（Move, Get, Pull 三选一）。

动作详解：

1. **Move (移动):**
   - 指令格式: `move <id> <dir>`
   - 方向: `0` (右, y+1), `1` (左, y-1), `2` (上, x-1), `3` (下, x+1)。
   - **碰撞规则 (简化版):**
     - 机器人不能移动出地图边界。
     - 机器人不能移动到 `*` 或 `#` 格子。
     - **互斥锁:** 机器人不能移动到 **本帧开始时已有其他机器人占据** 的格子。

        - *这意味着：即使机器人B在本帧从(0,1)移动走了，机器人A在本帧也不能移动到(0,1)。必须等下一帧该格子空出来。*
2. **Get (取货):**
    - 指令格式: `get <id>`
    - 条件:
        - 机器人当前位置 `(x, y)` 存在货物。
        - 机器人当前 `has_goods == 0`。
    - 效果: 货物从地图上消失，机器人 `has_goods` 变为 1 (携带该货物的价值)。
3. **Pull (卸货/装船):**
    - 指令格式: `pull <id>`
    - 条件:
        - 机器人位于泊位区域（地图字符为 `B`）。
        - 机器人当前 `has_goods == 1`。
    - 效果: 机器人 `has_goods` 变为 0，货物价值被累加到 **该泊位对应的虚拟轮船** 的货舱中。

4.2 货物 (Goods)

- **生成:** 在空地随机生成。
- **属性:** 坐标 `(x, y)`，价值 `val` (10-100之间)。
- **消失:** 货物生成后若 **1000帧** 内未被拾取，会自动消失。

4.3 轮船 (Ship)

- **数量:** 固定 5 艘（ID: 0-4）。
- **机制 (简化版):**
    - 为了降低难度，本题目不模拟轮船的航行路径。
    - 我们假设地图上有 5 个泊位区域，每个区域绑定一艘轮船。
    - 当机器人执行 `pull` 时，货物会自动进入该泊位对应的轮船货舱。
- **Go (运输/卖货):**
    - 指令格式: `go <ship_id>`
    - 效果: 立即卖出该船上所有货物。
    - **收益公式:** `Total Money += sum(货物价值)`。
    - 货舱清空，船只状态重置。

---

5. 输入输出协议 (C++ 参考)

程序必须严格遵守以下格式，否则判题器将无法识别。

5.1 初始化 (仅一次)

无。程序启动后直接进入第1帧。你需要自行读取本地地图文件（如需）。

5.2 每一帧的输入 (从 stdin 读取)

Plaintext

```
<帧号 ID> <当前金钱 Money>
<场上货物数量 K>
<货物1_x> <货物1_y> <货物1_价值>
...
<货物K_x> <货物K_y> <货物K_价值>
<机器人0_是否带货> <机器人0_x> <机器人0_y> <机器人0_状态>
...
<机器人9_是否带货> <机器人9_x> <机器人9_y> <机器人9_状态>
<轮船0_状态> <轮船0_目标泊位>
...
<轮船4_状态> <轮船4_目标泊位>
OK
```

C++ 读取示例:

C++

```
int id, money, k;
cin >> id >> money >> k;
for(int i=0; i<k; ++i) { ... } // 读货物for(int i=0; i<10; ++i) { ... } // 读机器人for(int i=0; i<5; ++i) { ... } // 读轮船string end_str; cin >> end_str; // 吞掉 "OK"
```

5.3 每一帧的输出 (向 stdout 打印)

输出若干行指令，最后必须输出一行 `OK`。

Plaintext

```
move 0 1
```

```
get 1
move 2 3
pull 5
go 0
OK
```

注意：

1. **指令数量不限：** 可以给所有机器人发指令，也可以只给部分发。未收到指令的机器人维持原地不动。
2. **Flush：** C++ 中使用 `endl` 通常会自动刷新缓冲区。如果使用 `\n`，请务必调用 `fflush(stdout)` 或 `cout.flush()`，否则判题器会因收不到数据而卡死（超时）。

---

6. 评分与要求

6.1 评分公式

$$Score = Final\_Money$$

- 资金只在执行 `go` 指令卖货成功时增加。
- 机器人手里拿着的货物不算分。
- 放在船上但没执行 `go` 指令卖掉的货物不算分。

6.2 考核点

1. **算法设计文档 (30%)**
   - 描述你的路径规划算法 (BFS / A*)。
   - 解释你的任务调度策略（如何决定谁去捡哪个货物）。
   - 分析时间复杂度。
2. **代码实现能力 (40%)**
   - 代码风格良好，模块化设计。
   - 能够正确处理 IO，无运行时错误。
3. **性能与策略 (30%)**
   - 在相同的测试地图下，分数越高越好。
   - 解决了多机器人冲突问题（不发生死锁）。

---

7. 常见问题 (FAQ)

Q: 两个机器人面对面走会怎么样?

A: 假设 A 在 (0,0), B 在 (0,1)。A向右 move 0 0, B向左 move 1 1。

根据冲突规则：A的目标是(0,1)，该位置本帧由B占据 -> A移动失败，停在(0,0)。B的目标是(0,0)，该位置本帧由A占据 -> B移动失败，停在(0,1)。

结果：两人都动不了。 你需要写算法避免这种情况（例如让其中一个等待）。

Q: 我需要打印地图来调试吗?

A: 绝对不要向 stdout 打印调试信息。这会破坏交互协议，导致判题器读取错误的数据。请使用 cerr (标准错误输出) 来打印调试日志，判题器会将 stderr 的内容显示在控制台，但不会干扰数据流。

Q: 轮船ID和泊位怎么对应?

A: 在本简化版中，你可以简单地认为：

- 泊位区域被划分为几个簇。
- 当你执行 `pull` 时，判题器会自动把货加到默认的 0号船（或者你可以查看 `judge.py` 源码了解它是如何分配的）。
- 为了拿到分，建议每隔 100 帧对所有船只发送 `go` 指令：
- C++

代码块

```
1   for(int i=0; i<5; i++) cout << "go " << i << endl;
```

---

8. 开始指南

1. 运行 `gen_map.py` 生成地图。
2. 这是一份**完整、严谨且详细的**大作业任务说明书。
3. 编译你的 C++ 代码： `g++ main.cpp -o main` (Windows下是 main.exe)。
4. 运行判题器： `python judge.py` 。
5. 观察控制台输出的分数，修改算法，不断优化