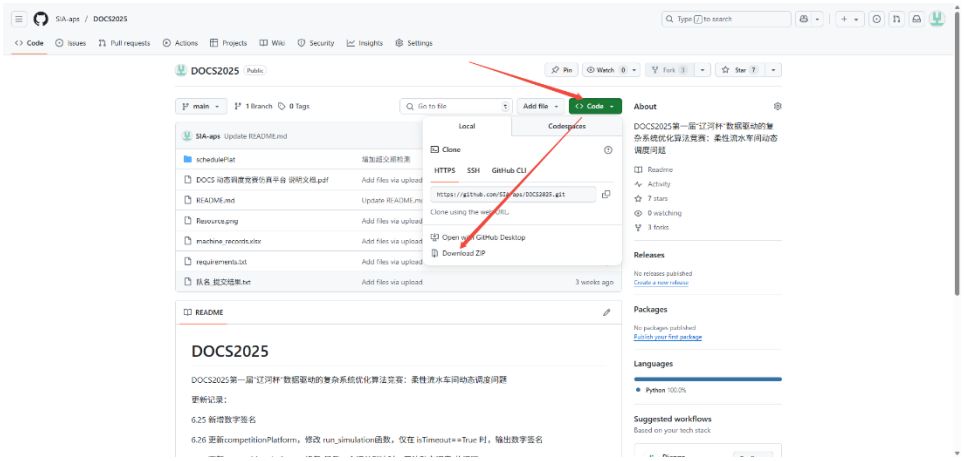
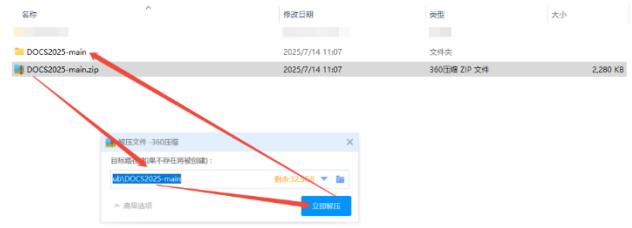
(一) 下载代码

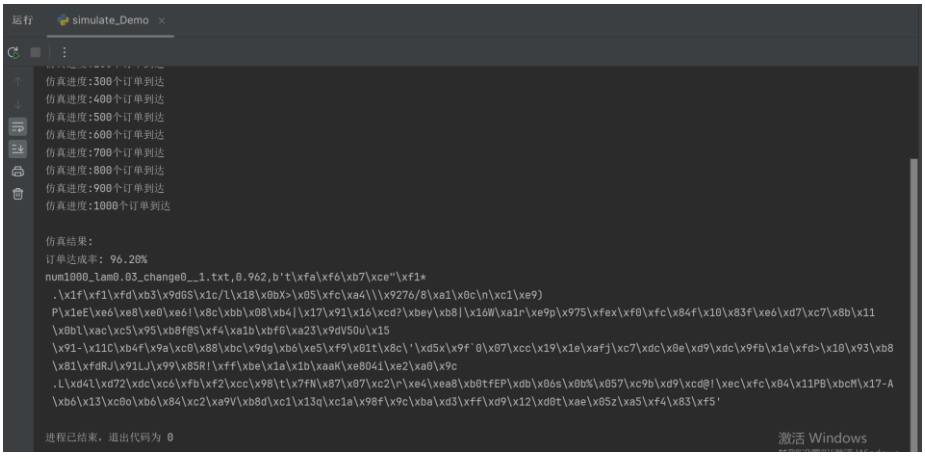
1. 登录 GitHub：https://github.com/SIA-aps/DOCS2025
   如果您无法登录网站，请联系赛事人员索要代码。
2. 点击“Code”，点击“Download ZIP”，下载代码



3. 解压，获取完整的工程文件



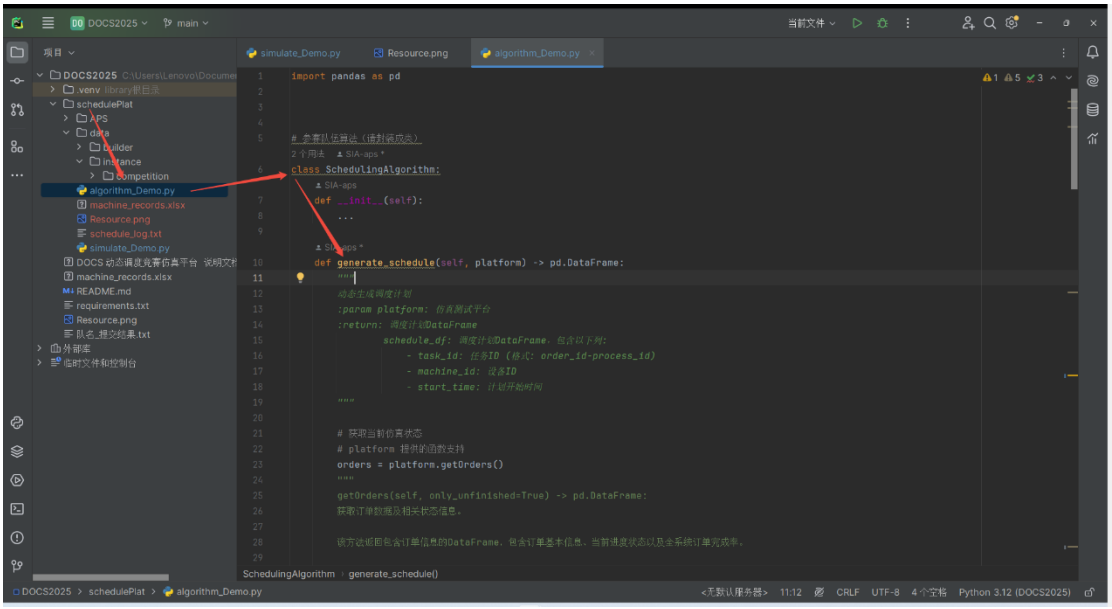
4. 运行
   在您的开发环境中，运行“schedulePlat”目录下的“simulate_Demo.py”。
   如果正常运行，说明您的代码已经正确下载。



注：这行代码显示错误是正常情况，不影响您的运行。

(二) 编写参赛代码

1. 在“schedulePlat”目录下的“algorithm_Demo.py”中编辑您的参赛代码。
   注：其他的后缀为.pyc 的代码，均为仿真平台代码，不需要您编辑。



2. 仿真平台原理
   仿真平台采用**事件驱动**的重调度架构，其核心流程如下：
   a) 事件触发
      当新订单动态到达时，平台将自动生成一个决策点（Decision Point）。
   b) 算法调用
      决策点触发后，平台立即调用参赛者算法类 SchedulingAlgorithm 中的 generate_schedule() 函数：

```
def generate_schedule(self, platform) -> pd.DataFrame:
    """
    动态生成调度计划
    :param platform: 仿真测试平台
    :return: 调度计划DataFrame
            schedule_df: 调度计划DataFrame，包含以下列:
                - task_id: 任务ID (格式: order_id-process_id)
                - machine_id: 设备ID
                - start_time: 计划开始时间
    """
```

   c) 计划生成与输出
      该函数需输出决策点后的重调度计划，以结构化 DataFrame 呈现，包含三列关键信息：
      task_id：任务唯一标识，格式为 M-order_id-process_id（例如：M-1-0，表示订单 1 的生产阶段 0 的任务）。
      machine_id：设备资源分配 ID
      start_time：任务的计划开始时间。

3. **可能用到的函数**：
   在您开发参赛代码的过程中，可能会需要调度下列由仿真平台提供的函数：

a) 当前订单信息：

```
getOrders(self, only_unfinished=True) -> pd.DataFrame
获取订单数据及相关状态信息。
    该方法返回包含订单信息的 DataFrame，包含订单基本信息、当前进度状态以及全系统订单完成率。
    Args:
        only_unfinished (bool, optional): 是否只返回未完成的订单。默认为 True。
            - True: 仅返回尚未完成的订单
            - False: 返回所有订单（包括已完成的）
    Returns:
        pd.DataFrame: 包含订单信息的 DataFrame，包含以下列：
            - order_id: 订单唯一标识
            - product_type: 产品类型
            - arrival_time: 订单到达时间
            - due_date: 订单交期
            - current_stage: 当前处理工序（如果当前没有进行中的工序则为上一个工序的结果。若首工序也未开始，则为 None）
            - assigned_machine: 分配到的机器 ID（如果当前没有分配则为为上一个工序的分配结果。若首工序也未开始，则为 None）
            - start_time: 当前工序开始时间（如果未开始则为上一个工序的开始时间。若首工序也未开始，则为 None）
            - end_time: 当前工序结束时间（如果未开始则为上一个工序的结束时间。若首工序也未开始，则为 None）
            - fulfillment_rate: 当前系统订单达成率（已完成订单/所有已到达订单）
```

b) 当前机器状态：

```
getCurrentMachineStatus(self) -> pd.DataFrame
获取当前时刻所有机器的状态信息。
    该方法返回一个 DataFrame，描述在当前时间点各机器的状态（空闲或正在执行的任务信息）。
    Returns:
        pd.DataFrame: 包含每台机器当前状态的 DataFrame，包含以下列：
            - task_id: 当前执行的任务 ID（如果机器空闲则为 None）
            - start_time: 当前任务的开始时间（如果机器空闲则为 None）
            - end_time: 当前任务的结束时间（如果机器空闲则为 None）
```

c) 产品加工时间：

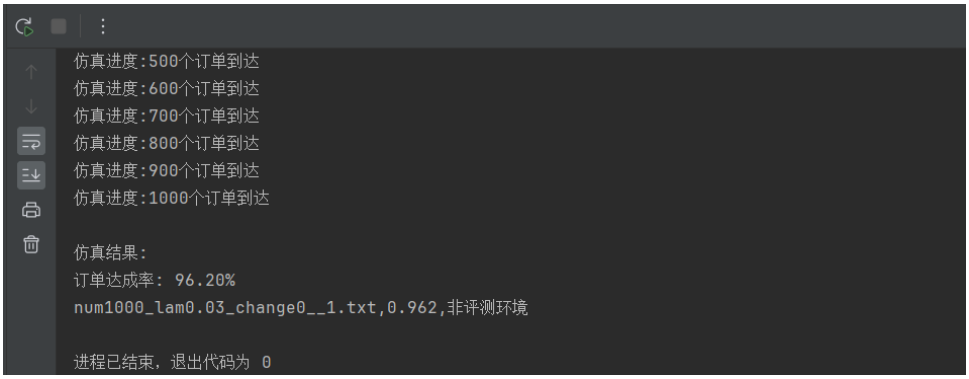
```
getMBOM(self) -> pd.DataFrame:
获取制造 BOM(Bill of Materials)信息。
    该方法从仿真实例中提取产品的工艺路线信息，包括每个产品在各生产阶段可选用的设备和相应处理时间。
    Returns:
        pd.DataFrame: 包含制造 BOM 信息的 DataFrame，包含以下列：
            - product_type: 产品类型标识（格式为"0"、"1"、"2"等）
            - stage: 生产阶段标识（格式为"0"、"1"、"2"等）
            - machine_id: 可用于该工序的设备 ID
            - process_time(s): 在该设备上完成该工序所需的处理时间（秒）
```

d) 当前时刻：

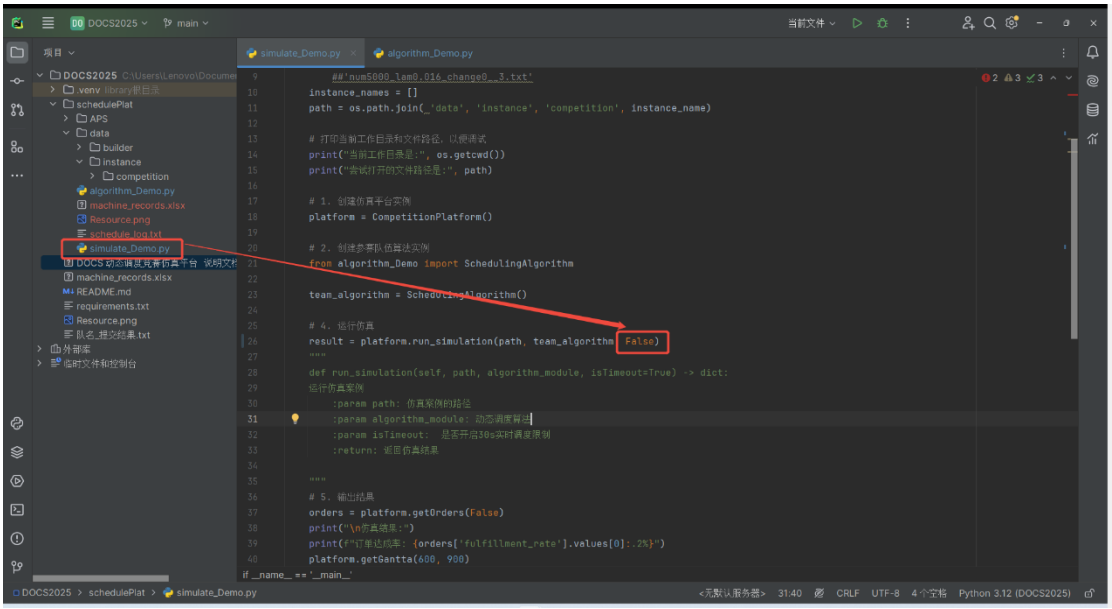
```
getSimulationTime(self) -> float:
获取当前仿真时间（从仿真开始起经过的时间）。
    该方法计算从仿真基础时间点（通常是仿真启动时间）到当前时刻经过的时间。
    Returns:
        float: 仿真经过的时间（以秒为单位）
```

4. Debug

当您完成参赛代码编写后，运行“schedulePlat”目录下的“simulate_Demo.py”。如果未出现报错，并输出订单达成率，则说明算法已经成功通过测试。

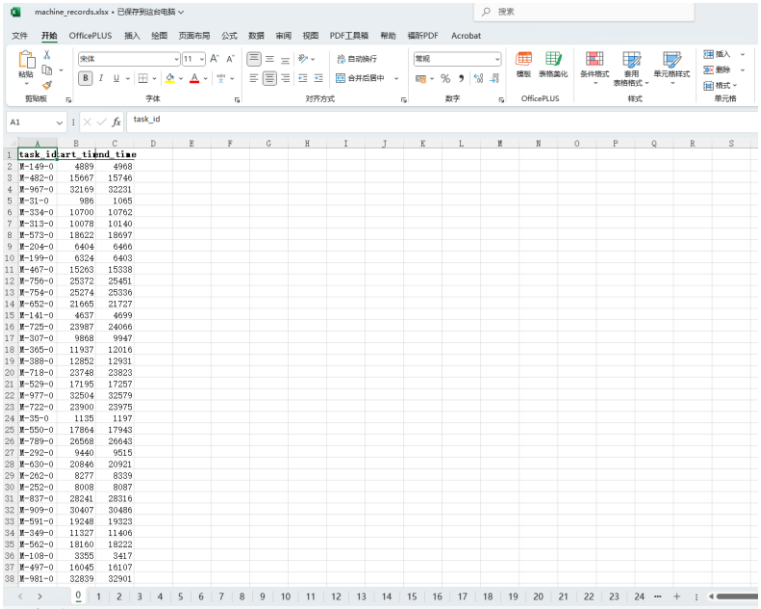


注意：在 debug 时，请将“simulate_Demo.py”中的“True”改为“False”，否则可能因为“超时”而报错。



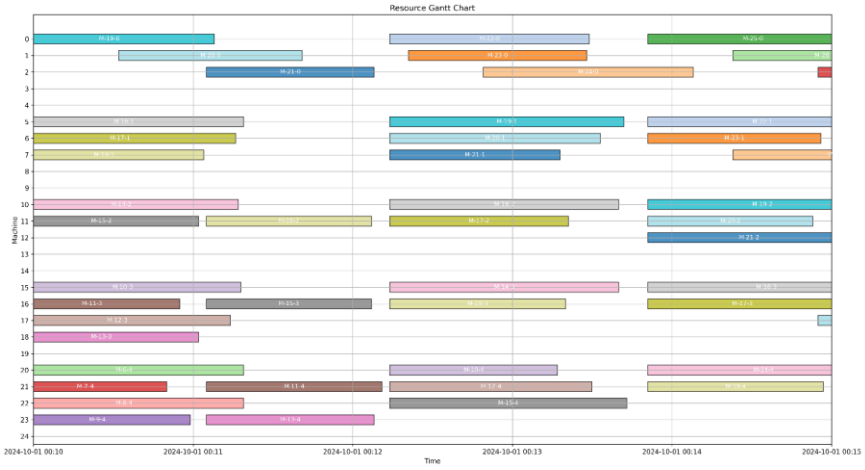
5. 更详细的结果查看

仿真平台输出了完整的作业计划，在“machine_records.xlsx”中。



其中，每个工作表记录了一台机器的完整作业记录。

此外，还可以查看某时段的机器甘特图。



通过调用 getGantta(self, startTime, endTime)函数实现。

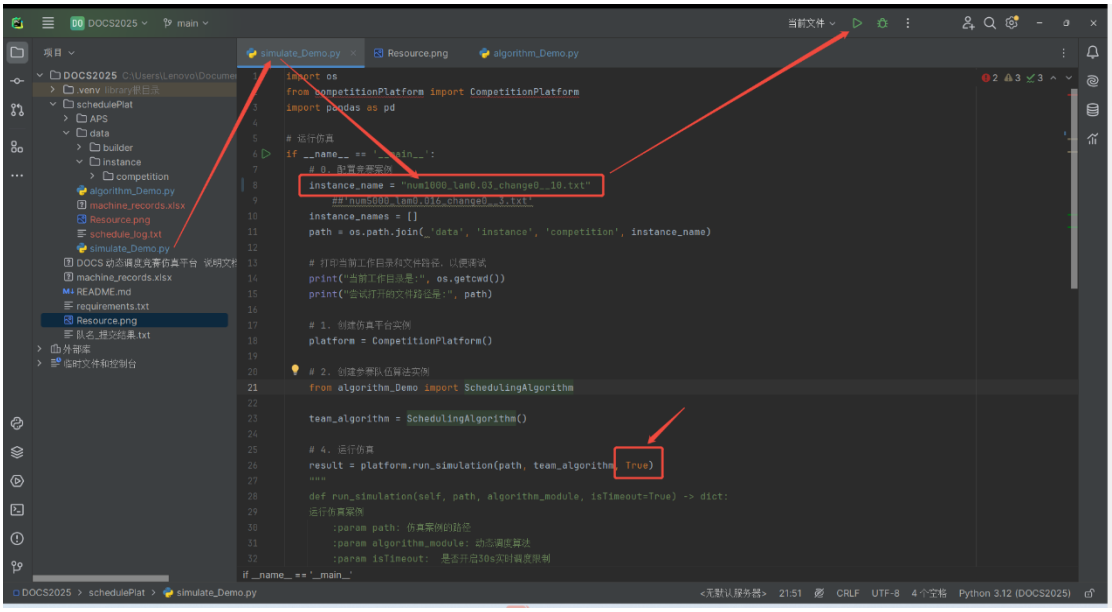
def getGantta(self, startTime, endTime)
生成甘特图 :param startTime: 开始时间点 :param endTime: 结束时间点

(三) 提交比赛结果

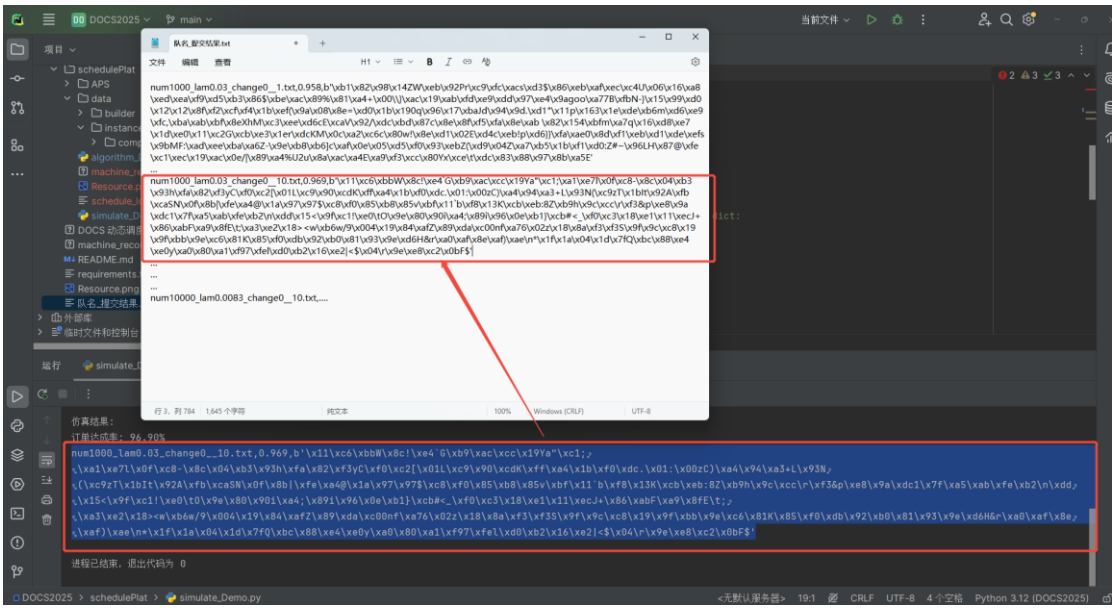
1. 在全部算例下进行仿真

   在“simulate_Demo.py”中，修改参数“instance_name”，配置不同的比赛实例，并运行。

   注意!!! 此时要将“simulate_Demo.py”中 platform.run_simulation 函数的参数“False”改为“True”



2. 将每个 instance 的输出结果，保存在“队名_提交结果.txt”中，将用于初赛的最终评审。



关键要求：

必须包含所有测试实例的结果（缺一不可）

实例顺序不限，但每行对应一个完整实例

不可修改签名内容（任何修改都将导致验证失败）

数字签名需保持原始格式（含开头的 b'和结尾的'）

3. 提交规范

A. 必交内容

- 主算法文件（队名_Algorithm.py）：必须包含完整的 SchedulingAlgorithm 类实现
- 说明文档（队名_doc.pdf）：简述算法思路和关键函数
- 依赖文件(requirements.txt)：列出非系统级 Python 库
- 初赛仿真结果（队名_提交结果.txt）：由竞赛仿真平台输出的结果，用于初赛评审。

B. 提交方式：

- 邮箱提交：
  - mail: wangrui@sia.cn
  - 提交压缩包命名：团队名_提交日期.zip

(四) 赛事人员联系方式

联系电话：18202411758

邮　　箱：wangrui@sia.cn

企业微信：