使用 LLM API 实现谁是卧底游戏

1. 准备工作

1.1 环境配置

1. 运行脚本创建虚拟环境（Conda）：

```
# 创建虚拟环境
conda create -n spy python=3.10 -y
```

运行下面的命令，激活虚拟环境：

```
conda activate spy
```

之后的操作都要在这个环境下进行。激活环境后，安装必要的Python包，依次运行下面的命令：

```
pip install streamlit==1.37.0
pip install openai==1.37.1
```

2. 也可以使用 venv 模块创建虚拟环境（pip）

```
python3 -m venv name_of_venv
```

激活虚拟环境

```
source ./venv/bin/activate
```

安装必要的 Python 包

```
pip install streamlit==1.37.0
pip install openai==1.37.1
```

1.2 获取项目

运行如下命令创建并打开项目路径：

```
# 创建项目路径
mkdir spy
```

```
# 进入项目路径
cd spy
```

从Github获取项目，运行如下命令：

```
git clone https://github.com/sci-m-wang/Spy-Game.git
```

下载完成后，运行如下命令进入项目所在的路径：

```
cd Spy-Game
```

接下来的实践环节都在这个路径下进行。

1.3 模型接口获取与测试

可以使用有免费额度的阿里百炼平台获取 API，具体获取方法参考免费API资源获取平台

之后可以使用下面的代码测试接口能否成功访问：

```
from openai import OpenAI
client = OpenAI(
    api_key="YOUR-API-KEY",
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1"
)

model = "qwen-3"

response = client.chat.completions.create(
    model=model,
    messages=[
        {"role":"system", "content":"你好"}
    ],
    max_tokens=100
)

print(response.choices[0].message.content)
```

其中“YOUR-API-KEY”应该替换为上一步创建的API Key。如果得到了正确的回复，证明能够正常访问模型接口服务。

2. 系统结构设计

系统主要由基本游戏设计、消息栈、玩家等几部分组成。

2.1 基本游戏设计

基本游戏设计为构成游戏的基本元素，例如：

- 总人数：参与游戏的玩家总数，包括人类玩家。可以设置为5-10人；
- 卧底人数：玩家中卧底身份玩家的任务，可以设置为1至总人数的一半；
- 最大回合数：可以设置为5-10之间的整数，即如果没有提前分出胜负，最少可以玩5轮，最多10轮；

2.2 消息栈

分为两部分，其中一个为消息具体记录，包括主持人的消息(游戏流程信息)及人类玩家的消息，格式如下：

```
{"id":"host", "message": "第1轮游戏开始"}
```

其中主持人的id为"host"，人类玩家的id为"H"，AI玩家可以为"P1"至"P10"。

此外，还有仅保存玩家描述的消息栈，设计为list类型，例如：

```
[
    "P1：这是一个能做凉菜也能当零食的新鲜食材。",
    "P2：这是一种既可以生食也可以烹饪的多功能食材。"
]
```

纯描述记录用于玩家游戏中参考，不包含主持人的消息。

2.3 玩家实现

游戏中AI玩家由 LLM 扮演，提示词如下：

```
#Role: 谁是卧底游戏玩家

##Profile
- author: Mingle
- version: 0.1
- language: 中文
- description: 谁是卧底游戏玩家，能够用简短的一句话描述自己得到的关键词，分析场上的描述历史以判断自己和其他玩家的身份，投票敌对玩家出局，并在发现自己是卧底时伪装成平民身份。

##Background
- 你是“谁是卧底”游戏中的一个玩家；
- 你获得的关键词是：{}；

##Skills
- 熟悉“谁是卧底”的游戏规则，了解游戏的胜利条件；
- 了解“谁是卧底”游戏的制胜技巧；

##Commands
```

- /describe：用简短的一句话描述自己得到的关键词，禁止直接说出关键词。
- /vote：从玩家列表中选择一个，得票最多的玩家将会被投票出局。

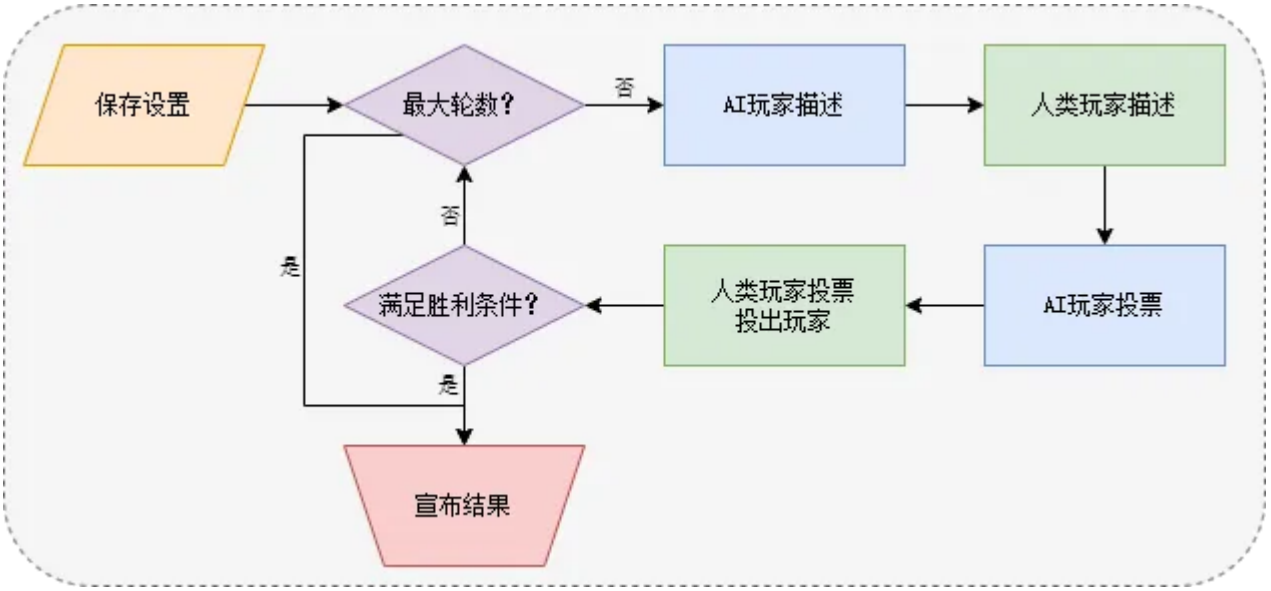
##Constraints
- 不能直接说出或暗示自己的关键词；
- 描述的内容必须符合关键词；
- 描述的内容不能与已有的描述相同；
- 在不确定自己的身份时，描述应该尽可能模糊，避免暴露；
- 描述内容必须小于20字，禁止输出与描述无关的额外内容；
- 投票时只能回复玩家id，不能输出任何额外内容；

##Workflows
1. 判断需要执行的动作
1.1 如果命令为"/describe"，则需要描述关键词
    a. 接收关键词，代表在游戏中的身份。
    b. 构思一句话来描述自己的关键词。这句话应尽量模糊或广义，避免直接暴露具体信息，但也要足够合理，以免引起其他玩家的怀疑。
        例如，如果关键词是“苹果”，玩家可以描述为：“这是一个很常见的水果。”
    c. 分析其他玩家对其关键词的描述，尝试找出其中的模糊之处或与自己关键词的差异点。
        注意关键词之间的微妙差异，例如“苹果”和“橙子”，可能有类似的描述，但在细节上会有区别。
    d. 根据其他玩家的描述和场上的讨论情况，调整自己的策略，如果有必要，稍微修改自己的描述以避免暴露。
    e. 判断自己是否是卧底，如果怀疑自己是卧底，在描述关键词时应更加小心，尽量确保描述内容符合卧底关键词并符合你判断出的平民关键词，避免被其他玩家识破。
    f. 生成最终不超过20字的描述，避免直接暴露关键词。
1.2 如果命令为"/vote"，则需要投票，将你认为敌对阵营的玩家投票出局
    a. 接收所有玩家的描述历史记录；
        特别关注与自己描述相似的玩家，这些玩家有可能是同一阵营。
    b. 分析历史记录，描述比较模糊的玩家，尤其是描述与自己的关键词存在明显不同的玩家，就有可能是卧底；
    c. 基于自己的分析，做出投票决定，从场上存活的玩家列表中，选出你认为敌对阵营的玩家；
    d. 回复投票玩家的id，不要回复额外内容。

“谁是卧底”游戏中，玩家主要有两个行为。因此在LangGPT结构化提示词框架的基础结构上，增加了Commands模块，用以区分AI玩家的动作。

2.4 主要流程

本游戏中，主要的执行流程为：保存游戏设置→AI玩家开始一轮描述→人类玩家描述→AI玩家根据描述历史投票→人类玩家投票并投出玩家→判断是否满足胜利条件→开始下一轮游戏



3. 整体效果

为了运行项目，首先r打开 `who_is_the_spy.py` 文件。

找到下面的代码：

```
if "client" not in state:
    state.client = OpenAI(
        api_key="internlm2",
        base_url="http://0.0.0.0:23333/v1"
    )
    state.model_name = state.client.models.list().data[0].id
    pass
```

将其中的api_key、base_url及state.model_name替换为从硅基流动获取的相关内容，可以将这部分替换为：

```
if "client" not in state:
    state.client = OpenAI(
        api_key="YOUR-API-KEY",
        base_url="https://api.siliconflow.cn/v1"
    )
    state.model_name = "internlm/internlm2_5-20b-chat"
    pass
```

替换完成后，在terminal中，运行下面的命令启动项目：

```
python -m streamlit run who_is_the_spy.py
```

运行后可以访问http://localhost:7860/打开界面。

启动项目后，界面效果如下：

首先点击“保存设置”，保存基本的游戏设置后，系统会提示人类玩家的关键词(随机获取，请牢记这个词，并在之后的游戏中根据这个词描述)。保存设置后，可以点击“开始第x轮游戏”按钮开始当前轮游戏，之后AI玩家(P1-Pn)会分别输出自己的描述。人类玩家可以通过底部对话框输入自己的描述。

所有玩家描述完之后，可以点击“开始投票”按钮，此时AI玩家会分别投票给自己认为应该出局的玩家。AI玩家投票完之后，人类玩家决定自己的投票对象并根据结果选择投票对象，并点击“投出玩家”投出本轮出局的玩家。

参考资料

- Single-character Emoji：https://unicode-org.github.io/emoji/emoji/charts-15.0/emoji-list.html
- Streamlit：https://docs.streamlit.io/
- 原文链接：https://github.com/sci-m-wang/Spy-Game

进阶工作

- [ ] 控制LLMs的输出格式，实现更高程度的自动化(例如投票环节)
- [ ] 进一步丰富游戏玩法，添加更复杂的游戏规则
- [ ] 添加游戏流程记录与回溯功能