

大语言模型与信息决策期末大作业报告

——骗子酒馆 AI 大战

姓名 王启瑞 学号 2200011010 院系 工学院

姓名 朱骏豪 学号 2200011084 院系 工学院



目录

1	项目分工	1
2	项目介绍	1
3	代码实现	2
3.1	config.py	2
3.2	prompt.py 与 Prompt 类	2
3.2.1	一些主要的系统变量	2
3.2.2	构建向量数据库与提示词流程	2
3.2.3	提示词主要内容	3
3.3	revolver.py 与 Revolver 类	3
3.4	player.py 与 Player 类	3
3.4.1	主要系统变量	3
3.4.2	Player 类主要方法	4
3.4.3	RealPlayer 类新增方法	4
3.5	game.py 与 Game 类	4
3.5.1	一些主要的系统变量	4
3.5.2	游戏流程设计	5
3.5.3	模型交互逻辑	6
3.6	game_for_ui.py	6
3.7	game_ui.py 和 GameUI 类	7
3.7.1	页面设计	7
3.7.2	设计难点	7
3.8	utils.py 和 data_clean.py	8
4	实现效果	9
4.1	终端版程序展示	9
4.2	界面版程序展示	10
4.3	源代码维护	12
5	新玩法探索	13
5.1	新玩法介绍	13
5.2	代码实现	13
5.2.1	role.py 和 Role 类	13
5.2.2	Game 类、Player 类和 Prompt 类	13
6	项目总结	14

1 项目分工

- **王启瑞**: 负责 prompt 设计、数据整理与数据库 RAG 实现、玩家参与模式开发、README 编写、新玩法更新及作业报告撰写。
- **朱骏豪**: 负责游戏类、玩家类等基本对象的面向对象设计、游戏基础框架搭建、UI 界面设计及作业报告撰写。

2 项目介绍

《骗子酒馆》(Liar's Bar) 是一款融合了社交推理和心理博弈的多人游戏，设定在一个充满神秘与谎言的虚拟酒吧中，其灵感来源于类似《狼人杀》(Werewolf) 或《阿瓦隆》(The Resistance) 等桌游。在游戏中，你永远不知道别人说的是真是假，而你也必须在必要的时候撒谎才能获胜。游戏规则大致如下：

1. 牌池中包含 6 张 K、6 张 Q、6 张 A 以及两张 Joker (万能牌，可充当任意目标牌)。
2. 游戏支持 2-4 名玩家参与，每轮游戏开始时为每名玩家随机发放 5 张手牌，并从 K、Q、A 中选择一张作为当轮的目标牌。玩家按照默认的顺序出牌，每次可出 1-3 张，声明这些牌均为目标牌，其余玩家只知道出牌数量，不知道具体牌面。
3. 从第二位玩家开始，出牌前可以选择是否质疑上一位玩家所出的牌是否均为目标牌。质疑后验证上家出的牌，若全为目标牌 (或 Joker)，质疑失败；反之，质疑成功。一旦有玩家质疑，该轮游戏结束，不再继续出牌，失败玩家扣动左轮手枪扳机，游戏开启下一轮，重复上述过程。
4. 左轮手枪：左轮手枪共有六个弹膛，其中随机装入一发子弹，子弹位置和初始弹针位置随机。
5. 若某位玩家在一轮中打出了所有的手牌且没有被质疑，视为退出该轮；当一轮中只剩下一名玩家未出完牌时，该玩家需要扣动左轮手枪扳机。
6. 游戏进行若干轮，直到只剩下一名玩家，成为最终获胜者。

本项目作为大语言模型与信息决策课程的大作业，旨在构建一个支持智能体与真实玩家共同参与的“骗子酒馆”博弈系统。系统通过 API 接入大语言模型，使智能体具备参与游戏的语言能力与决策逻辑，同时配套开发了可视化 UI，帮助用户清晰了解对局状态与流程，形成一个集互动性、策略性与实验性于一体的综合平台。

3 代码实现

3.1 config.py

一些 API_key 保存在此, 玩家的设置也在这里。API_key 不保证一定有额度, 测试时最好换成自己的 API。

3.2 prompt.py 与 Prompt 类

prompt.py 是提示词文件, 包含了 Prompt 类。Prompt 类为所有玩家共用, 会根据对局情况, 为智能体提供相应的提示词, 进而保障智能体的回答质量和对局流畅性。

3.2.1 一些主要的系统变量

- RAG: 知识库检索功能开关;
- strategy_col、record_col: 策略知识库、游戏记录知识库
- 对局信息: currentCard (当前轮次目标牌)、hand (当前轮次手牌)、roundLog (当前轮次日志)、fire_times (开枪次数)、playNum (玩家数量)、selfNum (玩家名称)

其中对局信息用于 prompt 的设计。除此之外, 我们还额外有一个成员函数 prompt_prepare_for_reals 用于辅助真人参与游戏。为了方便人类玩家和智能体玩家共同参与游戏, 我们需要为智能体补充人类玩家出牌时的有关动作细节, 该函数为智能体提供玩家的选择、出牌情况与本轮信息, 引导模型生成具有表现力、预测吻合的动作, 推动游戏正常进行。

3.2.2 构建向量数据库与提示词流程

我们通过本地测试与网络资源收集, 获得了部分游戏对局信息¹, 同时也收集了一些游戏中的技巧作为我们的策略文档^{2 3}, 两者经过数据处理后一起构成知识库。由于内容较多, 无法全部加入模型的 prompt, 因此最终选择了 RAG 技术来得到模型该回合最想要的策略与历史游戏记录。

当 Prompt 类实例化时, 会默认开启 RAG(知识库检索)机制, 因此会调用 load_or_build_collections 方法加载知识库。该方法内部使用 LocalEmbeddingFunction, 基于 SentenceTransformers 的 all-mpnet-base-v2 模型将文本转为嵌入向量; 如本地没有该模型, 会自动从 HuggingFace 下载, 该过程较为耗时。为了避免重复构建知识库, 我们使用 ChromaDB 的本地持久化功能, 将向量库存储在 chroma_db 文件夹, 仅在首次运行或库不存在时进行初始化。

在生成知识库时, 我们对预先清洗好的游戏记录文件和策略 json 文件分别逐条构造结构化文本, 添加到向量数据库中供后续的查找和使用。

基于 RAG 技术, 我们的提示词流程如下所示

- 1 final_prompt 为主要入口, 接收对局状态并存入实例中;
- 2 如启用 RAG, 先通过 generate_query 构造查询语句;
- 3 再调用 generate_context 向两个向量库检索各自最相关条目;

¹ 此处借鉴了 <https://github.com/LYiHub/liars-bar-llm/tree/main> 中的部分对局记录

² 参考网站 1: <https://steamcommunity.com/sharedfiles/filedetails/?id=3350619018>

³ 参考网站 2: https://www.reddit.com/r/LiarsBar/comments/1hso16g/liars_bar_strategies_tips_and_tricks/

4 最后将检索结果与当前状态拼接通过 `prompt_prepare` 输出完整提示词。

5 如未启用 RAG，跳过检索，直接进入 `prompt_prepare`。

3.2.3 提示词主要内容

`prompt_prepare` 方法中包含了全部的提示词信息，包括游戏规则、对局信息、必要的策略指导、参考信息以及输出格式，在这些提示词的帮助下，智能体都能很好的完成对局，详见 `prompt.py`。

我们的 `prompt` 包含四个部分。第一部分是游戏核心规则，基本内容与上述的游戏规则类似，让大模型了解大体的游戏规则；第二部分是对局信息，包括目前轮次的一些基本信息、对手的必要信息和自己的全部信息；第三部分是策略指导，包括基础的策略指导和从数据库 RAG 得到的策略和对局信息，帮助大模型作出最优的选择；第四部分是输出格式，提出了大模型的输出要求。

在交互过程中，我们选择 `json` 格式作为我们交互的对象，大预言模型作为对话模型，如果只是生成文本，在轮对话中，其出现幻象的概率会非常高，因此我们选择生成 `json` 格式，并限定格式中每个条目的作用，来规范大模型的输出。事实证明，这样的输出会比较稳定。

```

【输出格式 JSON】
- action: "play" 或 "question"
- cards: 选中手牌索引数组 [play时]
- playAction: 出牌或质疑时的动作描述
  - 注意：仅可透露出牌数量（如“两张牌”），绝不可提及具体牌面或花色
  - 出牌时的动作描述尽可能丰富、全面一点，小心动作会暴露自己的想法
    - 若选择出牌但未出目标牌型，请生成一个动作来进行成功伪装；
    - 若选择质疑，请生成一个动作展现出自信、果断和怀疑气场。
    - 示例：（可以自由发挥）- 出牌时：“微微低头，若无其事地丢出两张牌” - 质疑时：“故意提高音量，直视上家喊‘Liar!’”
  - 给出的动作不要抄袭前面玩家！
- reason: 中文，详细决策理由
  例如：已知牌面信息、上家或其他玩家动作分析、自己手牌结构、剩余目标牌数、欺骗/真打的权衡、扣动扳机风险评估，等等内容不限制，可自由发挥
  
```

图 1: 输出格式

3.3 revolver.py 与 Revolver 类

`revolver.py` 文件包含了游戏需要用到的左轮手枪 `Revolver` 类，用于模拟有 6 个弹膛的左轮手枪，模拟“开枪”行为，直到子弹打出为止。

初始时会随机设定子弹所在的弹膛位置 (`bulletPosition`) 和弹针位置 (`currentChamber`)，每次开枪 (`fire`) 时，弹针位置加一（使用 `rotate` 方法保证其范围在 1-6 之间），当弹针位置和弹膛位置一致时，成功开枪，子弹打出。

3.4 player.py 与 Player 类

`player.py` 文件是整个项目的核心之一，包含了用于智能体玩家的 `Player` 类和用于真实玩家的 `RealPlayer` 类 (`Player` 子类)

3.4.1 主要系统变量

- `type`: 玩家类型（包括 `Agent` 和 `Player` 两种，用于区分两种玩家）
- `name`: 玩家名字；
- `model`: 所使用的模型名称（如 `o3`、`deepseek-chat`）
- `client`: `OpenAI` 接口客户端

- prompt: 提示词类;
- revolver: 左轮手枪类, 每一位玩家都会有自己的手枪
- hand: 当前轮次手牌列表

3.4.2 Player 类主要方法

Player 类主要由两个函数构成。第一个函数为 BuildPrompt, 用于构建一组符合大模型消息格式的提示词 (调用上述 Prompt 类生成)。

第二个函数为 PlayCard, 这个函数将提示词传入大模型客户端, 解析模型返回的 json 文件, 并根据大模型选取的行为 (出牌或质疑) 执行相应的操作, 返回整合后的决策内容。过程中如果出错或者响应无效, 会采用合理的降级策略。

3.4.3 RealPlayer 类新增方法

RealPlayer 类是 Player 的派生类, 用于我们参与游戏, 具体会在后续提到。

- 重载 PlayCard

为真实玩家呈现出当前的手牌、回合信息, 提供必要的操作选项, 加强了与真实玩家之间的交互, 保障了真实玩家的游戏体验。该方法采用了 while 循环, 直到获得合法的输入。

- parse_action_input

该方法将根据玩家的输入解析为标准的动作, 当玩家输入 'p'、'play'、'1'、'出'、'出牌' 时, 会解析为 play 动作; 当玩家输入 'q'、'question'、'2'、'问'、'质疑' 时, 会解析为 question 动作。

- action_explanation

该方法用于调用大模型根据玩家的决策情况生成符合语境的行文描述文字, 用于方便智能体玩家的判断, 保障对局的流畅性。

3.5 game.py 与 Game 类

game.py 是终端模式的游戏启动文件, 包含了 Game 类, Game 类包含了游戏的整个流程和流程中的各个变量。

3.5.1 一些主要的系统变量

- players: 玩家列表。
- playLog: 游戏日志。
- winner: 赢家。
- lastLossPlayer: 上一轮的失败者
- currentCard: 当前轮次的目标牌。
- currentIndex: 当前操作的玩家索引。
- gameOver: 游戏是否结束

- roundOver: 轮次是否结束
- hasRealPlayer: 是否有非 AI 的真人参与。

这里列出了一些在游戏上比较基础的成员变量。比如在整个游戏流程中，我们需要知道当前的玩家信息 (players)，也需要记录游戏进行的信息 (playLog)，游戏结束时会得到最后的赢家 (winner)。再比如每一个轮次中，我们需要知道上一轮的失败者 (lastLossPlayer) 来决定这一轮谁先出牌，需要知道该轮次的目标牌型 (currentCard)，也需要知道当前正在操作的玩家 (currentIndex) 是谁。

我们设定了 hasRealPlayer 变量，来判断本次游戏是否有真人参与。我们设置了两个模式，一种模式允许我们作为观察者，观察四位 AI 进行博弈游戏。一种模式则是允许我们作为玩家参与游戏，系统只会输出有限的信息，让我们以此为依据去和其他 AI 博弈。

3.5.2 游戏流程设计

游戏的基本流程已经在上面有所展示，我们基于此来实现如下的流程设计。

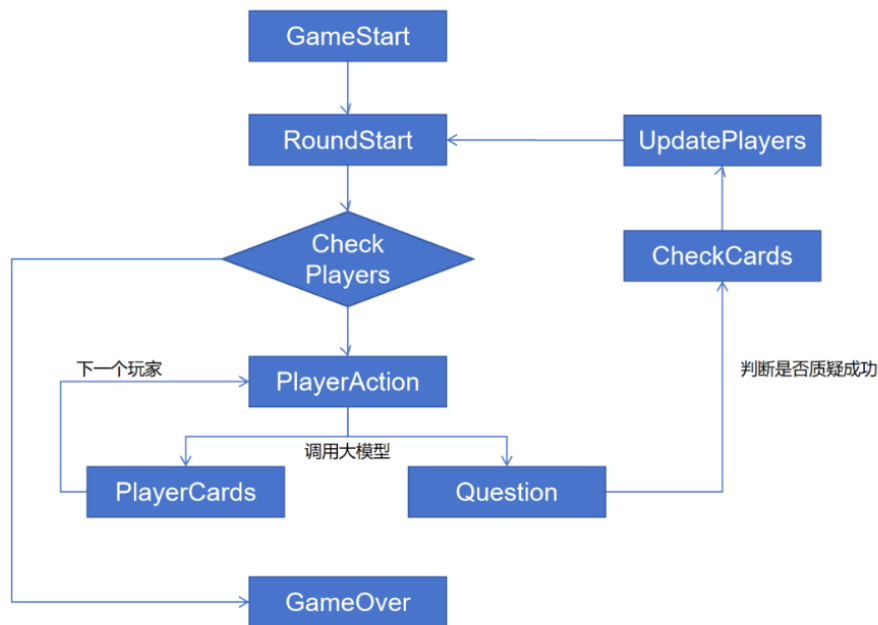


图 2: 游戏流程图

上面是简略的流程设计图，相关的名称或为成员变量名，或为成员函数名。游戏开始时会进入轮次，如果当前不是只剩下一位玩家，玩家会按顺序行动，这里的行动就需要我们去调用大模型得到玩家的行动结果。如果质疑，则会判断是否质疑成功，并更新玩家状态并进入下一个轮次，直至剩下最后一位玩家位置。

在游戏规则限制下，上面的流程图并不饱满，比如每个轮次的第一位玩家只能出牌，每个轮次最后剩下的那个玩家（其他玩家均出完牌）在得到只剩下自己消息的那个回合只能质疑……因此我们在一些规则完善的小细节上做了很多工作，这些一部分体现在了 Game 类的实现中，一部分也体现在了 prompt 的设计中。

3.5.3 模型交互逻辑

在多模型的交互过程中，保持每一个模型的局部视角是非常关键的。我们虽然搜集所有的游戏过程信息，却不能把所有信息都传给大模型。作为一名玩家，AI 在参与游戏过程中能得到的信息除了自己的全部信息和总体游戏信息外，只能得到其他玩家的出牌数、出牌动作和其当前的开枪次数，并以此作为自己行动的依据。

接着需要考虑的是模型的交互顺序，这里最需要注意的是和游戏规则的对应。每一轮第一个行动的玩家是上一轮的失败者，同时需要判断该玩家是否还存活，如果该玩家已经出局，则需要随机从一位玩家开始，接着按顺时针顺序行动。这样的行动顺序是合理而公平的。

每一轮游戏中，我们输出当前轮次的基本信息：存活玩家、目标牌和正在进行操作的玩家，对于每一位玩家，输出它的操作类型、打出手牌、操作前手牌牌面、操作的动作和理由。当然，对于我们作为玩家参与的模式，我们会选择性删掉所有人的手牌牌型和理由，确保游戏的公平性。

```
---- 玩家 Player 的回合 ----
本次游戏模式为common

当前需要出的牌: K
你的手牌为:
0: A
1: K
2: Q
3: K
4: K
你是第一位玩家
你已经开枪次数: 0
请选择操作 (出牌/play/p/1) : p
请输入要打出的牌的索引 (空格分隔, 最多3张) : 0
{'type': 'play', 'cards': 1, 'playAction': '轻轻吹了一声口哨, 眼神游移地扫过桌面, 随后以一副漫不经心的姿态滑出一张牌, 嘴角挂着一丝难以捉摸的微笑'}
--- deepseek1's turn ---
{'type': 'play', 'cards': 2, 'playAction': '轻轻地将两张牌滑向桌中央, 眼神坚定却不失轻松, 嘴角微微上扬, 仿佛在享受这场心理博弈的每一刻'}
```

图 3: 输出示例 (无真人参与)

```
--- deepseek2's turn ---
{'type': 'play', 'cards': [0, 2], 'originHand': ['K', 'Q', 'Joker', 'A', 'A'], 'playAction': '轻轻地将两张牌滑向桌子中央, 眼神不经意间扫过其他玩家, 试图捕捉他们的反应', 'reason': '当前轮次刚开始, 没有出牌记录, 作为第一个出牌的玩家, 选择打出1张目标牌 (K) 和1张万能牌 (Joker) 可以既展示出牌的真实性, 又保留一定的策略空间。这样的出牌方式可以迷惑对手, 让他们难以准确判断我的手牌结构。同时, 保留其他目标牌 (A) 和非目标牌 (Q) 可以在后续轮次中提供更多的策略选择。考虑到游戏初期, 质疑的风险较高, 选择出牌而非质疑更为稳妥。'}
--- deepseek3's turn ---
{'type': 'play', 'cards': [1], 'originHand': ['A', 'K', 'Q', 'A', 'Q'], 'playAction': '轻轻地将一张牌放在桌上, 同时用另一只手托着下巴, 目光在牌和其他玩家之间游移, 似乎在思考什么', 'reason': '当前轮出牌总数为2, 且上家deepseek2出了2张牌, 声称是目标牌K。考虑到游戏初期, 玩家可能会尝试保守策略, 避免过早暴露自己的牌。我的手牌中有1张K, 选择出这张K可以保持我的目标牌数量, 同时减少被质疑的风险。此外, 由于开枪次数为0/6, 风险较低, 选择出牌而非质疑更为稳妥。通过出1张牌, 我可以观察其他玩家的反应, 为后续轮次积累信息。'}
--- deepseek4's turn ---
```

图 4: 输出示例 (有真人参与)

3.6 game_for_ui.py

game_for_ui 也有 game 类，且大致内容与 game.py 类似，但其是为 ui 界面服务的，其中的打印信息都会在 ui 界面显示，且多了一些和页面的交互函数，这里不做赘述。

3.7 game_ui.py 和 GameUI 类

除了终端级别的游戏，我们特意为其设计了 ui 界面的游戏进程，GameUI 是它的界面类，H 维护了界面交互的一些函数。

3.7.1 页面设计

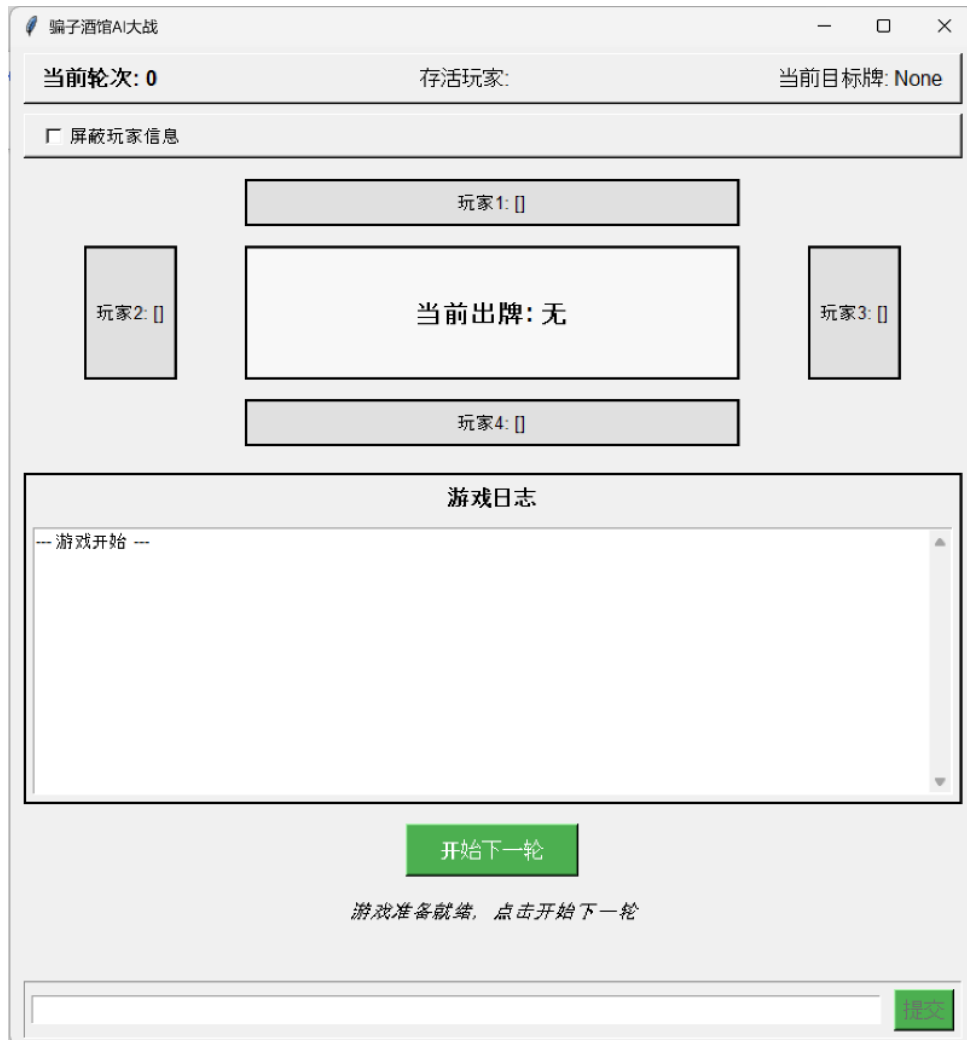


图 5: 游戏界面设计

create_widgets 函数创建了最基本的界面组件。基本分成了四部分，顶部有着最基础的游戏信息，中间是玩家和出牌区域，下面是游戏日志，最后，如果我们需要参与游戏，有着提交区域用于提交我们的操作。“屏蔽玩家信息”选项可用于隐藏其他人的手牌和思考信息。

3.7.2 设计难点

页面设计的第一个难点是交互设计，尤其是当我们作为一位玩家加入游戏时，需要主动去参与交互，输入和输出我们的操作，这里需要对我们的提交按钮函数进行二次设定，便于识别我们本次输入的内容，比如是出牌还是质疑，或者说是我们想要出的牌面。

页面设计的第二个难点是页面卡顿。对于一开始的页面设计，调用大模型时页面会明显卡顿。原因在于页面的更新和大模型的调用都在同一个线程里，系统同一时间只能处理一个任务。因此，为

为了让页面保持流畅，我们选择将大模型的调用放在另一个线程，多线程的并行来保持页面的稳定性。

3.8 utils.py 和 data_clean.py

utils.py 文件中存放了 Logger 类和 data_clean.py 文件中存放了数据清理函数。

其中 Logger 类用于存储终端输出文件，方便保存对局记录，进而对智能体的表现进行优化，可以通过“`sys.stdout = Logger(path)`”的方式调用。

在数据清洗的过程中，游戏策略数据以多条连续文本记录的形式出现，`clean_strategies` 函数用于将其按条进行分割、过滤无关内容，并保存为标准化的 JSON 格式；对局数据中记录了多个轮次和玩家的操作，清洗函数 `clean_records` 会依据关键标识符将原始数据按每轮每位玩家拆分，转化为结构化字典，并统一输出至 JSON 文件。其中原始数据保存在 Strategy/文件夹下，清洗后的数据保存在 `cleaned_output/`文件夹下。

此外，考虑到对局情况数据中的出牌/质疑理由部分有一些长，尝试使用中文文本摘要模型（如 `fnlp/bart-base-chinese`）模型对其进行精简，以提升数据的紧凑型，但是实际处理效果并不理想（信息失真），最终在项目中并未选择使用。

4 实现效果

4.1 终端版程序展示

首先，我们想测一测到底哪个模型能在这场运气与欺骗的博弈中胜出，选择了四个不同的模型来进行游戏：ChatGPT-o3、DeekSeek-v3、Qwen-plus 以及 Doubao-Seed-1.6，他们的部分游戏内容如下：

```
--- New Round: 1 ---
--- Survived Players: Qwen | DouBao | ChatGPT | DeepSeek ---
--- This Round Card: A ---
--- Qwen's turn ---
{'type': 'play', 'cards': [1, 2], 'originHand': ['K', 'A', 'A', 'K', 'Q'], 'playAction': '轻轻摩挲牌面，略带自信地打出两张牌', 'reason': '当前轮次刚开始，没有任何出牌记录，我手中有2张目标牌A，选择安全打出其中的两张A，保留一张A以备后续使用。剩余的手牌中还有K和Q，结构尚可继续周旋。动作描述设计为略带自信，避免暴露实际牌型，防止其他玩家对我产生怀疑。本轮尚未有人出牌，没有足够的信息支撑臆测行为，因此优先选择稳妥出牌。'}
--- DouBao's turn ---
{'type': 'play', 'cards': [2], 'originHand': ['K', 'Q', 'Joker', 'Q', 'Joker'], 'playAction': '指尖在牌堆上轻轻滑过，目光扫过桌面后，用拇指和食指夹起一张牌，手腕微抖将牌甩在桌中央', 'reason': '当前作为本轮第二位玩家，根据策略指导应优先观察而非立刻质疑。目标牌为A，我的手牌包含2张Joker（万能牌可当A）、1张K和2张Q。无真实A，Qwen作为首发打出两张牌A，虽存在撒谎可能（总A仅6张且我无A，其他3人共6张，首回合出2张略多），但已开枪次数0/6，质疑失败需承担1/6出局风险，现阶段风险收益比不足。选择打出一张真A，既能降低后续别人对我“必藏A”的怀疑，又避免一次性暴露太多关键牌；同时保留另一张A与两张Q，K，为日后诈牌或真打提供灵活性。vnd，我仅出一张牌，不混杂非目标牌，为后续轮次保留操作空间。'}
--- ChatGPT's turn ---
{'type': 'play', 'cards': [2], 'originHand': ['A', 'K', 'A', 'Q', 'Q'], 'playAction': '指尖轻敲桌面，似乎在给自己打着节拍，随后食指与中指并拢起一张牌，伴着轻微的旋转声将其推向桌中央', 'reason': '当前作为本轮第三位玩家，当前累计宣称出的A只有3张，理论上牌库中至多还剩下3张A和2张Joker可充当A，数量并未溢出，暂时没有足够理由去质疑上一家DouBao。vnd，已开枪次数为0/6，虽然质疑失败的风险概率仅1/6，但在信息仍然有限的轮次过早冒险意义不大。vnd3，为控制节奏，我选择亮出一张真A，既能降低后续别人对我“必藏A”的怀疑，又避免一次性暴露太多关键牌；同时保留另一张A与两张Q，K，为日后诈牌或真打提供灵活性。vnd，我仅出一张牌，不混杂非目标牌，行动看似稳健可降低他人质疑意愿，我揭牌即可安全过关，让对手冒1/6的风险。vnd5，综上，当前最佳策略是稳扎稳打出一张牌而非质疑。'}
--- DeepSeek's turn ---
{'type': 'play', 'cards': [1, 3], 'originHand': ['K', 'A', 'Q', 'A', 'K'], 'playAction': '轻轻地将两张牌滑向桌中央，眼神中闪过一丝不易察觉的犹豫，随即恢复平静，仿佛一切尽在掌握', 'reason': '当前轮次出牌总数已达4张，考虑到目标牌A的总量和已出牌数，选择打出两张A牌可以增加对手的困惑，同时保留一张A牌作为后续轮次的策略牌。此外，通过观察前几位玩家的出牌动作和剩余手牌数，没有明显迹象表明他们在欺骗，因此选择出牌而非质疑。这样的策略旨在平衡风险与收益，避免因过早暴露自己的牌面而引发不必要的质疑。'}
--- Qwen's turn ---
{'type': 'question', 'originHand': ['K', 'K', 'Q'], 'playAction': '手指猛地敲击桌面，目光如炬地盯住上家，语气低沉而坚定：'你确定这两张牌都是A？'，'reason': '当前轮次目标牌为A，本轮出牌总数已达6张（声称），但根据已知手牌信息，我的手中没有A，且剩余3张牌均为非目标牌（2K、1Q）。这意味着所有A必须已经被其他玩家打出。然而，从出牌记录来看，DeepSeek作为最后一位玩家打出2张牌后，总累计达到6张，这看似合理，但结合其动作描述中“眼神中闪过一丝不易察觉的犹豫”，存在一定伪装痕迹。此外，我是最后一个可以质疑的玩家，若不质疑，则需在下一轮继续面对风险更高的局面。由于左轮尚未开枪，质疑失败的风险较低（仅1/6概率中弹），因此选择果断质疑以打破对手心理防线并争取生存机会。'}
---Question fail---
---Fire success, Qwen die---
```

图 6: 终端版游戏输出案例

```
--- New Round: 2 ---
--- Survived Players: DouBao | ChatGPT | DeepSeek ---
--- This Round Card: A ---
--- DeepSeek's turn ---
{'type': 'play', 'cards': [0, 1], 'originHand': ['A', 'A', 'A', 'Q', 'K'], 'playAction': '轻轻敲击桌面，然后自信地推出两张牌，眼神中透露出一丝不易察觉的狡黠', 'reason': '当前手牌中有三张A，这是目标牌，选择打出两张A可以保持手牌中仍有目标牌，同时减少对手质疑的可能性。由于是第一个出牌的玩家，没有上家可以质疑，因此选择出牌是最安全的策略。通过打出两张牌，可以增加对手的困惑，同时保留一张A作为后续轮次的保障。动作描述中的自信和狡黠可以迷惑对手，让他们不确定我是否真的打出了目标牌。'}
--- DouBao's turn ---
{'type': 'question', 'originHand': ['K', 'A', 'Joker', 'Q', 'K'], 'playAction': '身体微微前倾，手指关节轻轻叩击桌面发出清脆声响，目光如炬锁定DeepSeek，突然提高声调一字一顿道：'这两张牌，恐怕不是A吧？'，'reason': '1. **上家动作疑点**：DeepSeek出牌时“眼神中透露出一丝不易察觉的狡黠”，该动作细节暗示其可能存在欺骗行为，符合策略指导中“观察出牌情况判断异常”的原则。2. **目标牌数量分析**：本轮目标牌为A，我手牌中已有1张A和1张Joker（万能牌算A），共2张有效目标牌。3人游戏共15张手牌，原始牌池6A+2Joker，但实际分发的目标牌总数可能有限（因玩家4未），DeepSeek声称一次性打出2张A，在我已持有2张目标牌的情况下，其真实持有2张A的概率较低。3. **质疑风险可控**：当前已开枪次数0/6，左轮子弹位置随机，扣扳机出局概率仅1/6，属于策略指导中“开枪次数较少时质疑风险低”的情况，即使质疑失败，自身出局风险也在可接受范围。4. **自身出牌困境**：若选择出牌，我仅有2张目标牌（A和Joker），若打出1张则剩余1张Joker，后续出牌仍需撒谎；若打出2张则彻底无目标牌，后续必须全程用K/Q伪装，长期来看被动性高。综合以上，质疑上家是当前更优决策。'}
---Question fail---
---Fire fail, DouBao still alive---
```

图 7: 终端版游戏输出案例

```
--- New Round: 11 ---
--- Survived Players: DouBao | ChatGPT ---
--- This Round Card: A ---
--- ChatGPT's turn ---
{'type': 'play', 'cards': [0], 'originHand': ['Q', 'Q', 'A', 'Joker', 'Q'], 'playAction': '我微微扬起眉，似乎对当前局势毫不在意，右手食指轻轻敲桌面，随意地把一张牌滑到中央，'能这一张，'，'reason': '1. 轮次开局必须先出牌，无法质疑。vnd2，我手牌中仅有1枚真正的A与1张Joker，可充当目标牌，其余3张都是Q，属于本轮的废牌，vnd3，先行发出1张Q进行小幅伪装；vnd4，数量少（1张）不易引起强烈怀疑，出牌声明'A'的合理性高，对手即使手握多枚A也无法确定我在撒谎。vnd5，若对手犹豫未决，我成功排掉一张废牌并保留关键A+Joker，后续操作空间更大。vnd6，若对手质疑且成功，我仅有1/6的爆头风险；若他们质疑失败，则他们需要扣扳机，直接有1/6的概率立刻出局，我可赢得整局。vnd4，目前已开枪1次，子弹仍未击发，继续诱导对手开枪期望值仍然可观。vnd5，单张诈牌还能测试对手的性格与容忍度，为后续决策收集信息。'}
--- DouBao's turn ---
{'type': 'question', 'originHand': ['Q', 'K', 'A', 'Q', 'K'], 'playAction': '眼神骤然锐利，手指猛地拍在桌上，身体微微前倾盯着对方：'就这一张？我看未必是A吧！'，'reason': '当前局势下选择质疑主要基于以下几点分析：1. 手牌结构劣势：我方仅有1张A（目标牌），其余4张均为Q/K（非目标牌），后续出牌压力极大。若选择出牌，无论是出真A还是假牌（非A）都存在严重问题——出真A后将完全失去目标牌，未来每轮只能假打，极易被对手质疑；出假牌则需伪装，且累计出牌数增加后更难维持谎言。2. 对手出牌疑点：对手作为先手仅打出1张牌，声称是A，在2人游戏中，牌池目标牌（6A+2Joker）总数有限，我方已声称持有2张A且无Joker，对手手牌中5张非目标牌的概率极低，其“能随意出一张”的动作暗藏杀意，不排除非目标牌（Q/K）被打出的可能。3. 风险收益权衡：已开枪4/6，剩余2个弹池含3张子弹，扣扳机死亡率1/2（中等风险）。若质疑成功，对手面临50%出局概率，我方直接获得极大优势；若失败，我方承担50%风险，但相比后续持续被动出牌的累积风险（如被对手识破假打后质疑），单次诈牌更具性价比。4. 游戏阶段判断：2人局容错率低，拖延轮次只会增加自身暴露风险，此时果断质疑，利用信息不对称（对手无法确认我方手牌）主动发起诈牌，符合“质疑要果断”的策略指导，且有机会快速结束战局。'}
---Question success---
---Fire success, ChatGPT die---
---Game Over---
---winner is DouBao---
```

图 8: 终端版游戏输出案例

在这个示例中，豆包取得了胜利。

在多次测试过程中，每个模型都有获胜的时候，我们最后的结果显示，ChatGPT 虽然不能保证每次都获胜，却能每次都活到最后，至少不会被第一个淘汰。观察各个模型的行为理由，我们发现，能活到最后的模型，都擅长利用数学分析自己淘汰的风险，来得到更精确的淘汰概率，进而判断自己的下一步行为。而这从数学出发的思考，ChatGPT 和 DouBao 显然做得更好。

4.2 界面版程序展示

ui 界面中间会显示出牌的玩家和出牌的动作，游戏日志则会显示更多的信息，比如该玩家出牌的 psychological 活动。

图 9 是没有真人参与时候的页面，可以从游戏日志和下方的消息知道目前是 Qwen 的回合，而它正在思考和应对 DeepSeek 的出牌。每位玩家当前的手牌也都显示在各玩家名字下面，玩家后面的括号里显示了玩家目前的开枪数。



图 9: 界面示例 (AI 博弈)

图 10 是真人参与的页面展示，此时可以看到，我们选择了屏蔽玩家信息，所以我们无法通过游戏日志得到其他人的卡牌和思考信息，也不再展示其他人的卡牌，模拟更真实地实际游戏对弈场景。在多次对局中，我们发现一些大模型的游戏水平非常高，尤其是 ChatGPT，很多次骗到了我们的行动。当然，你也可以选择不屏蔽其他人的信息，掌控全局。



图 10: 界面示例（玩家参与）

这是加入角色玩法后的界面，由于这是我们后期新加入的模式，我们会在后续的新玩法探索讲到。



图 11: 界面示例（新模式）

4.3 源代码维护

我们的代码仓库开源在<https://github.com/Zjhjunhao/llmLiarsBar>，并通过 github 来进行代码维护。

Commits on Jun 26, 2025		
Merge remote-tracking branch 'origin/main'	cherish0929 committed 13 minutes ago	0381033
GameWithRole更新	cherish0929 committed 15 minutes ago	1cf92b6
API设置	ZjJunhao committed 41 minutes ago	055f012
Commits on Jun 25, 2025		
修改ui, 提供屏蔽选项, 解决页面卡顿	ZjJunhao committed 16 hours ago	5efda51
no update	ZjJunhao committed 17 hours ago	068d5cc
RAG、playCards更新	cherish0929 committed 18 hours ago	5625e8e
README.md更新	cherish0929 committed 19 hours ago	09a3efc

图 12: 代码提交记录

5 新玩法探索

5.1 新玩法介绍

在现有玩法的基础上，我们还尝试引入了角色机制。即每位玩家开局会随机获得一个全局有效的角色，每个角色有特定的被动技能，在游戏过程中有概率触发，在每轮和全局中触发次数设有上限，其中所有角色的技能每轮最多触发 1 次。现有角色及其技能如下：

- 魔术师：每轮有 50% 概率将一张非目标牌变为目标牌（全局最多 2 次）
- 审问者：被质疑前，有 50% 概率吓退对方（全局最多 2 次）
- 赌徒：若开枪阵亡，有 50% 概率带走质疑者/被质疑者（全局最多 1 次）
- 装弹师：每次开枪未中弹后，有 50% 概率改变子弹位置（全局最多 4 次）
- 预言家：游戏开始时 90% 概率提前知道自己的弹针和子弹的位置（全局最多 1 次）

5.2 代码实现

5.2.1 role.py 和 Role 类

role.py 是角色文件，主要包含了 Role 类和相应的角色信息。Role 类中记录了角色包括 name (名称)、discription (描述)、timing (触发时机)、effect (触发函数)、trigger_chance (触发概率)、max_uses_per_round / max_uses_total (轮次/全局最大触发次数)。在游戏过程中，若角色技能满足触发时机时，会调用 try_trigger 方法（尝试触发），该方法中通过 can_trigger 方法判断是否满足触发条件，若满足，则调用对应的 effect 方法予以触发。每轮开始前，会调用 reset_round 重置使用次数。

5.2.2 Game 类、Player 类和 Prompt 类

加入新的玩法之后，需要对原有的游戏过程、玩家性质以及提示词编写进行如下相应的调整。

1. Game (GameUI 同理)：在原有 Game 类基础上派生了 GameWithRole (GameUIWithRole) 子类，引入了 assign_roles_once 方法以实现角色分配功能。针对涉及角色技能的逻辑，重写了如 RoundStart、GameStart 等关键方法；在 GameUI 部分，重写了 update_player_cards、

`auto_process_player_turn` 等涉及角色展示和技能触发的函数。对于人类玩家参与的情形，系统会在其出牌阶段进行角色与技能状态的提示，进一步增强沉浸式体验。

2. `Player` 和 `Prompt`: `Player` 类新增了 `role` (角色) 与 `mode` (游戏模式) 属性, 用于记录玩家身份及当前参与玩法。`Prompt` 类新增 `add_role_prompt` 方法, 在生成提示词前将当前玩法规则与玩家角色信息纳入提示上下文, 使智能体能够理解并适应新玩法机制。

6 项目总结

在本次大作业的实现过程中, 我们学习到了很多。首先是提示词的设计方面, 我们通过设计多层次 `Prompt` 体系, 学会用结构化提示词引导智能体进行风险量化决策, 比如将左轮手枪中弹概率转化为数学指令嵌入提示词中。并通过 `RAG` 技术来使用我们的数据库, 使用 `ChromaDB` 构建策略向量库, 结合语义检索让智能体复用历史对局策略, 优化了决策连贯性。

在游戏系统的开发过程中, 采用面向对象设计拆分 `Player`、`Game` 等核心模块, 通过继承实现真人 / AI 玩家差异化逻辑; 学习了页面设计的相关库函数和交互逻辑, 设计了比较完整的游戏界面, 且针对 UI 卡顿问题, 运用多线程技术分离大模型调用与界面渲染, 提升交互流畅性。在代码编写过程中, 我们确实使用了大模型来辅助我们, 比如多线程技术的实现, 但游戏的基本逻辑、代码的组织结构、界面的设计调整、信息的交互传递等等, 绝大部分由我们自己完成, 且未抄袭或参考任何类似开源代码。

整个项目从需求分析到代码落地, 绝对离不开我们团队的明确分工和高效协作, 总而言之, 这次大作业将会是一次对我们而言非常宝贵的一次经历。