# 大语言模型大作业-骗子酒馆AI大战

王启瑞 朱骏豪

## 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，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，知道只剩下一名玩家，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。

## 二、代码实现

### 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。

### 2.prompt.py与 Prompt类

prompt.py是提示词文件，包含了Prompt类，Prompt类所有玩家共用，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。

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）

2.构建向量数据库

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，过程可能耗时。为了避免重复构建，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jsonl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。

3.提示词生成流程

* final\_prompt为主要入口，接收对局状态并存入实例中；
* 如启用 RAG，先通过 generate\_query构造查询语句；
* 再调用 generate\_context向两个向量库检索各自最相关条目；
* 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。
* 如未启用 RAG，跳过检索，直接进入 prompt\_prepare。

4.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（统一采用json格式输出），在这些提示词的帮助下，智能体都能很好的完成对局。

5.辅助动作生成（prompt\_prepare\_for\_reals）

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，预测了吻合的动作，推动游戏正常进行。

### 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。

### 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）

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

hand：当前轮次手牌列表

1. Player类主要方法

* BuildPrompt方法

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）

* PlayCard方法

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相应无效，会采用合理的降级策略。

1. RealPlayer类新增方法

* 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\_explaination

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。

### 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。

**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。

值得一提的是，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。

**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。

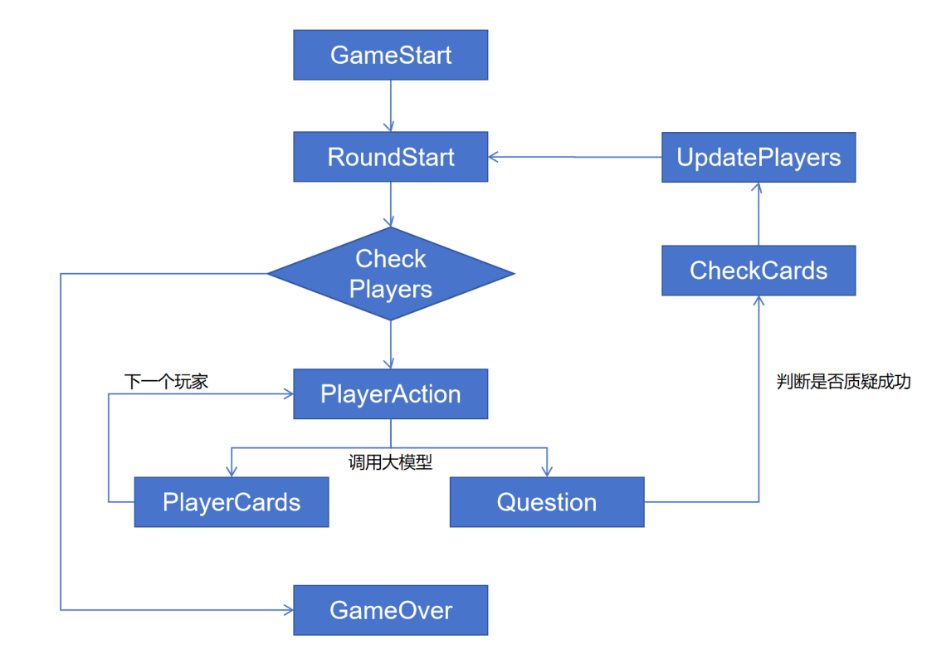


图 1游戏流程图

上面是简略的流程设计图，相关的名称或为成员变量命，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.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，这部分的逻辑在后续的player类会更详细的介绍。

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。

### 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。

### 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，维护了界面交互的一些函数。

**1.页面设计**



图 3 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。

**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。

### 8.utils.py

utils.py文件中存放了项目中用到的工具，包括Logger类和数据清理函数。

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。

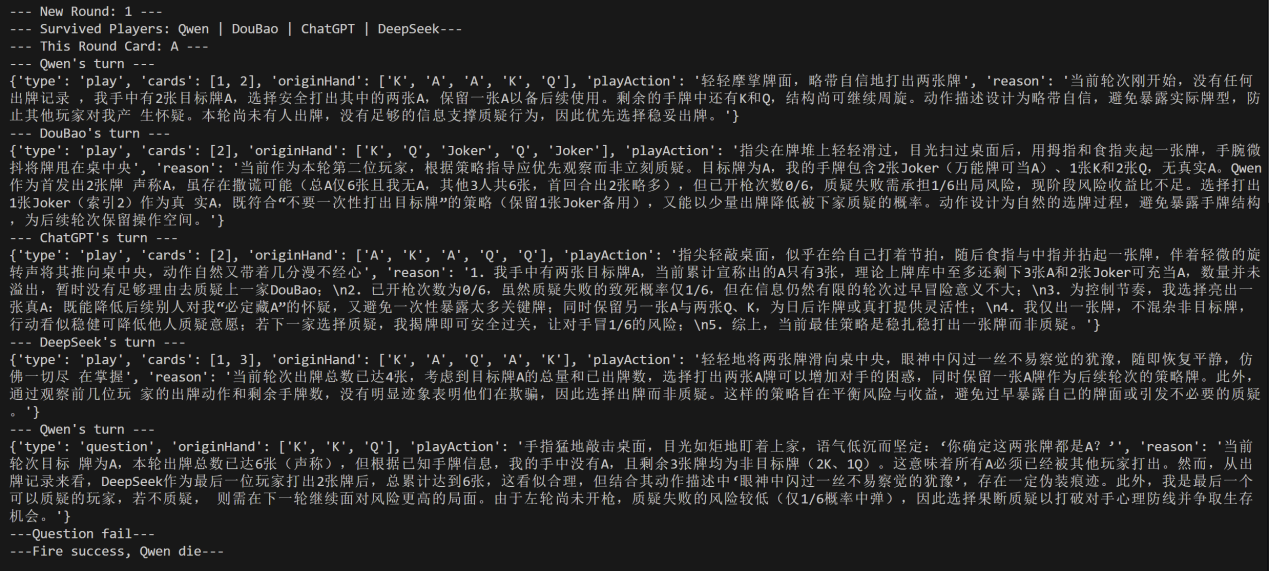
在数据清洗的过程中，游戏策略数据以多条连续文本记录的形式出现，clean\_strategies 函数用于将其按条进行分割、过滤无关内容，并保存为标准化的 JSONL 格式；对局数据中记录了多个轮次和玩家的操作，清洗函数clean\_records会依据关键标识符将原始数据按**每轮每位玩家**拆分，转化为结构化字典，并统一输出至 JSONL 文件。其中原始数据保存在Strategy/文件夹下，清洗后的数据保存在cleaned\_output/文件夹下。

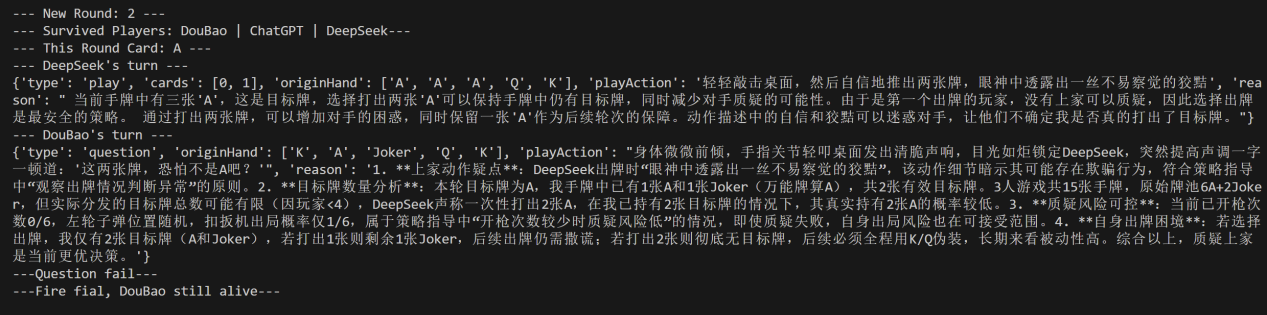
此外，考虑到对局情况数据中的出牌/质疑理由部分有一些长，尝试使用中文文本摘要模型（如fnlp/bart-base-chinese）模型对其进行精简，以提升数据的紧凑型，但是实际处理效果并不理想（信息失真），最终在项目中并未选择使用。

## 三、实现效果

### 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：





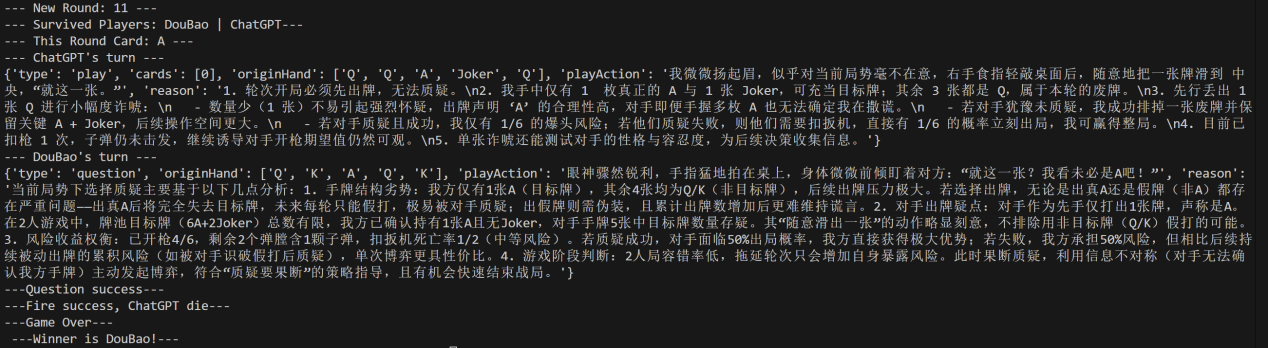


图 2 终端版游戏输出案例

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。观察各个模型的行动理由，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。

### 1. 界面版程序展示

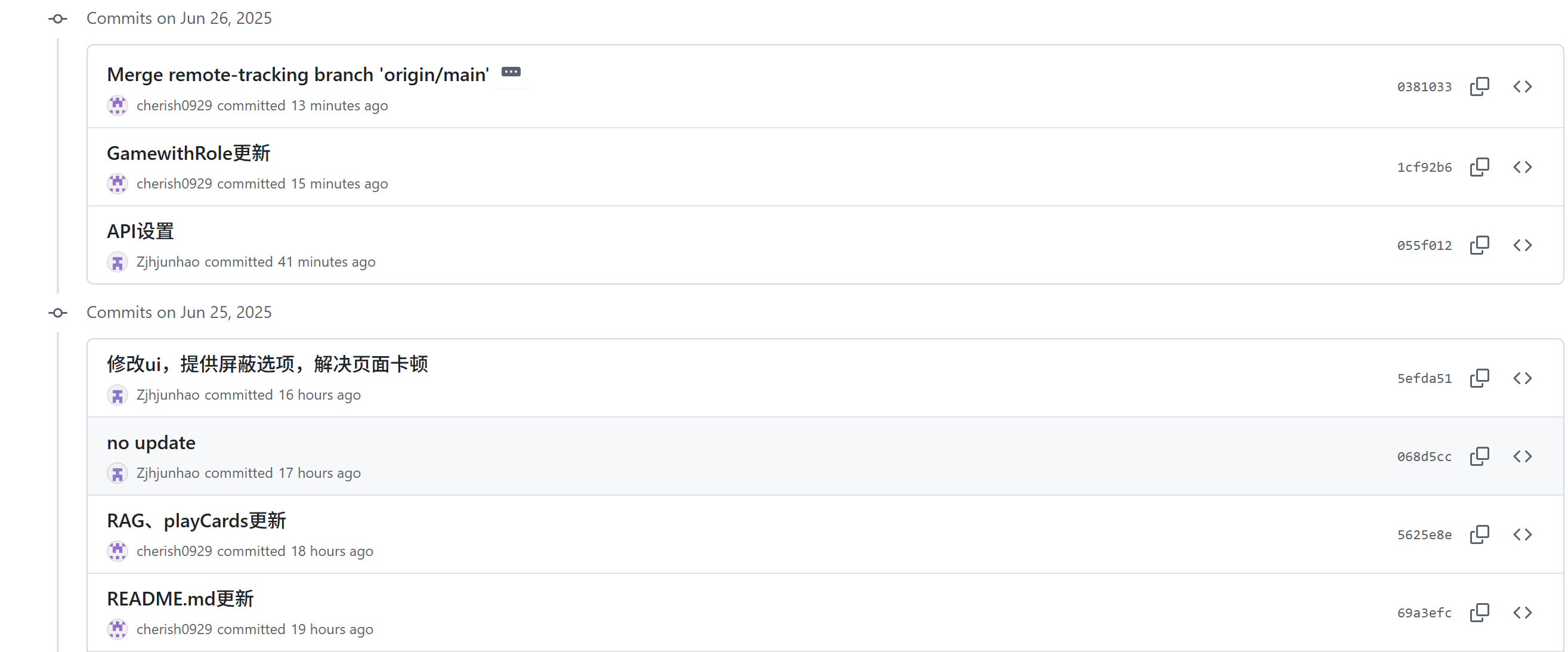


图 4 界面示例（AI博弈）



图 5界面示例（玩家参与）

我们的代码仓库开源在[Zjhjunhao/llmLiarsBar: 大语言模型大作业](https://github.com/Zjhjunhao/llmLiarsBar)，并通过github来进行代码维护。



## 四、新玩法探索