# 大语言模型大作业-骗子酒馆AI大战

王启瑞 朱骏豪

## 一、项目介绍

## 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.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。

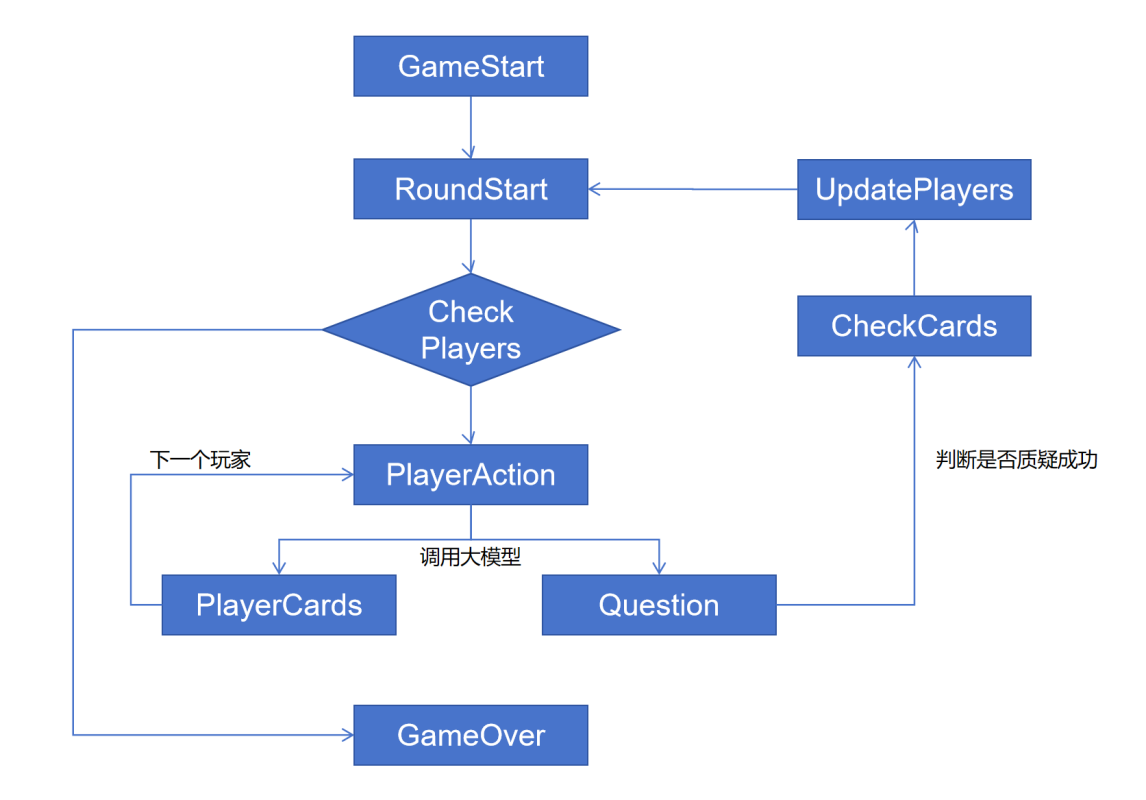


图 1游戏流程图

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。

### 3.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。

### 4.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**

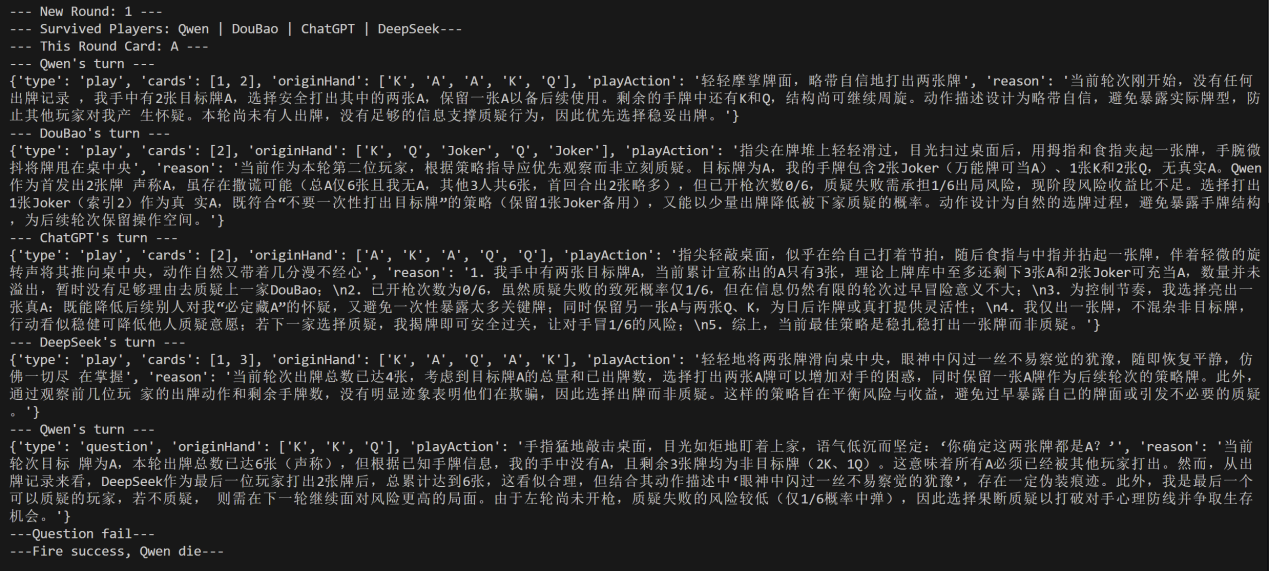
页面设计的第一个难点是交互设计，尤其是当我们作为一位玩家加入游戏时，需要主动去参与交互，输入和输出我们的操作，这里需要对我们的提交按钮函数进行二次设定，便于识别我们本次输入的内容，比如是出牌还是质疑，或者说是我们想要出的牌面。

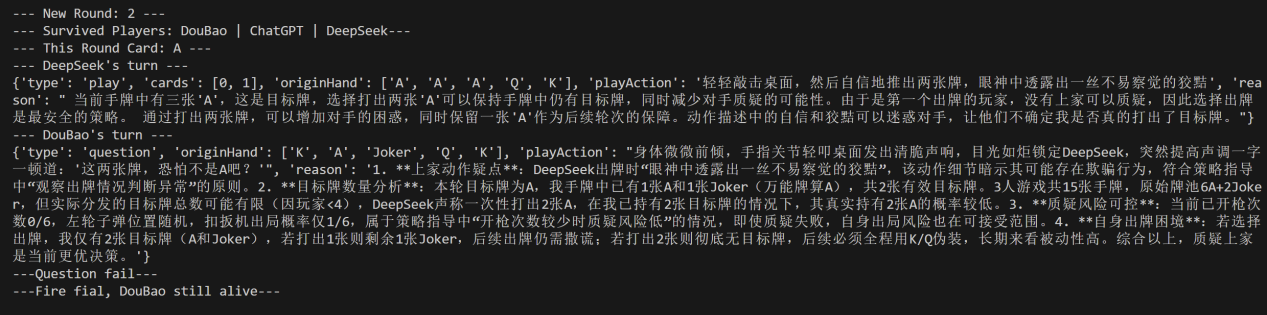
页面设计的第二个难点是页面卡顿。对于一开始的页面设计，调用大模型时页面会明显卡顿。原因在于页面的更新和大模型的调用都在同一个线程里，系统同一时间只能处理一个任务。因此，为了让页面保持流畅，我们选择将大模型的调用放在另一个线程，多线程的并行来保持页面的稳定性。

## 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：





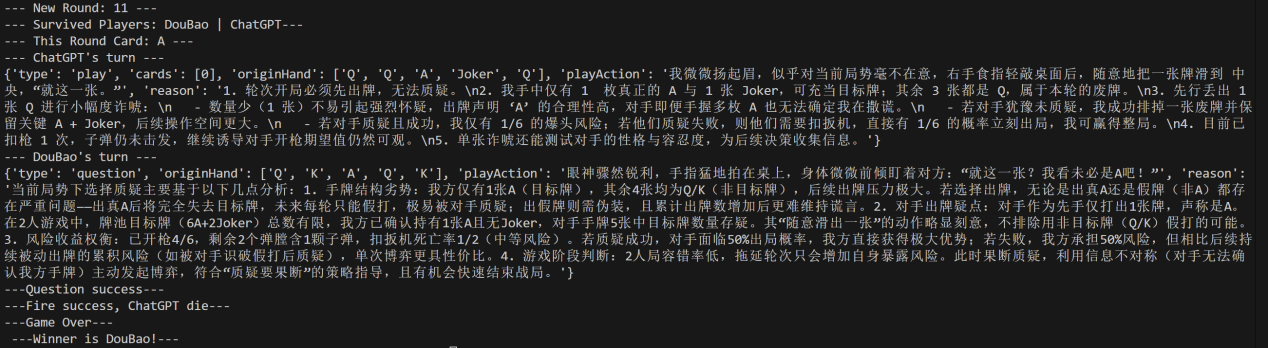


图 2 终端版游戏输出案例

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。

## 2.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。

