《控制论合作者》说明文档：
规则、自述及要求

目录

1 规则

1.1 配件

- 卡牌 50 张 (红、黄、蓝、绿、白色各 10 张，各包含 3 张数字 1、2 张数字 2 至 4、1 张数字 5)
- 提示指示物 8 枚
- 错误指示物 3 枚

1.2 概述

本游戏是一款 2-5 人的卡牌游戏，玩家需要合作将手中不同颜色的卡牌各自按数字升序打出。在游戏过程中，玩家不能查看自己的手牌信息，仅能通过玩家之间的相互提示得到部分信息。每个回合中，玩家必须选择并进行弃置 1 张手牌、打出 1 张手牌或提示其他玩家，直至卡组抽完且所有玩家又各进行了一回合、所有颜色数字 5 都已在场上或错误地打出了 3 张卡牌。

1.3 初始设置

玩家共同获得全部 8 枚提示指示物，将 3 枚错误指示物置于一旁。将所有卡牌洗混正面朝下构成卡组，玩家按玩家数正面朝外地抽取对应数量的卡牌 (下称“抽牌”) 作为手牌:

- 2 或 3 人：5 张
- 4 或 5 人：4 张

那之后，随机决定起始玩家。

1.4 游戏流程

自起始玩家起，所有玩家按顺时针方向依次进行回合，直至满足结束条件。在一位玩家的回合中，其必须选择并进行下述行动之一：

- 支付 1 枚提示指示物，选择另一位玩家，指出其手牌中一种颜色或数字的所有卡牌 (不可提示该玩家没有的颜色或数字，没有提示指示物时不可选择)。
- 获得 1 枚提示指示物，正面朝上弃置 1 张手牌，随后抽 1 张牌 (拥有全部 8 个提示指示物时不可选择，当卡组无卡可抽时不进行抽牌，下同)。
- 打出 1 张手牌，判断是否正确打出并相应地置于场上或弃置，随后抽 1 张牌。

1.5 打出手牌正确性判断

手牌仅在下述条件之一满足时是正确打出的：

- 数字为 1 且场上没有与该卡牌颜色相同的卡牌。
- 数字为场上与该卡牌同色的卡牌中最大的数字的后继数。

若手牌被正确打出，将其置于场上，否则弃置并获得 1 枚错误指示物。如果打出的手牌数字恰为 5 且玩家拥有的提示指示物数量不足 8 枚，获得 1 枚提示指示物。

1.6 游戏结束

当下述条件之一满足时，游戏立刻结束，进行终局计分：

- 玩家共计获得了 3 枚错误指示物。
- 所有颜色的数字 5 均在场上。
- 一位玩家需要抽牌而卡组无牌可抽，随后含该玩家在内的所有玩家均进行了 1 回合。

1.7 终局计分

玩家共同的分数为场上卡牌的数量。

1.8 玩家限制

除上述规则规定的回合行动外，玩家不能以其他形式进行交流，任何玩家指出另一位玩家的手牌时，目标玩家**必须**向前者重复以确认提示信息。

1.9 变体规则：彩虹

该部分规则仅在任务步骤 3 中需要考虑。

本变体规则引入了一种新的颜色，并为该颜色附加了特殊规则。

1.9.1 配件变化

额外增加了彩虹卡牌数字 1 共 3 张、数字 2 至 4 各 2 张及 1 张数字 5，合计 10 张。

1.9.2 游戏流程变化

玩家在回合中不能选择彩虹为提示的颜色，但提示任何其他颜色时，彩虹卡牌被视为具有该种颜色，故必须作为该种颜色的卡牌被提示。在打出彩虹卡牌时，仍然视为第六种颜色。

2 程序文件说明

2.1 文件清单

文件或文件夹名	概述
game.py	包含游戏类及游戏相关的若干类的定义
gameRainbow.py	将 game.py 修改为变体规则版本
basePlayer.py	包含玩家抽象类的定义
player.py	玩家类模板，通过控制台进行回合行动输入
playerRainbow.py	变体规则玩家类模板，通过控制台进行回合行动输入
remotePlayer.py	远程玩家类，游戏类触发监听器时将所有参数打包为 JSON，将其通过 websockets 转发，并将接收的玩家行动 JSON 转为行动类返回至游戏类
simpleServer.py	开启一个 websockets 服务器，指定数量客户端连接后开始游戏
requirements.txt	上述 python 脚本的库依赖
remoteReceiverPlayer	Vue 项目文件夹，simpleServer 的客户端。接收远程玩家类转发的 JSON，在 web 页面上显示结果并接收人类玩家回合行动输入

2.2 游戏及相关类定义

2.2.1 colors

卡牌颜色的枚举类，其中表示未知颜色、红色、黄色、蓝色、绿色、白色的整型值分别为-1 至 4。在 gameRainbow.py 中，表示彩虹的值为 5。

2.2.2 actions

玩家行动的枚举类，其中表示提示颜色、提示数字、打出卡牌、弃置卡牌的整型值分别为 0 至 3。

2.2.3 endgames

游戏结束原因的枚举类，其中表示错误指示物为 3、卡组为空且所有玩家均进行了最后的回合、所有颜色数字 5 均打出的整型值分别为 0 至 2。

2.2.4 invalidAction(Exception)

行动错误异常类，继承了 Exception 类。抛出该错误表示玩家输入的行动不符合规则。该类新增的成员变量包括：

- **message: str** 错误信息

重载及新增的成员方法包括：

- **___init___(message)**

构造方法，参数释义：

message: str 错误信息

- **___str___()**

  转为字符串时返回错误信息

2.2.5 card

卡牌类。该类的成员变量包括：

- **color: colors** 卡牌颜色
- **number: int** 卡牌数字

成员方法包括：

- **___init___(color,number)**

  构造方法，参数释义：

  color: colors 卡牌颜色

  number: int 卡牌数字

- **copy()**

  返回自身的深复制

2.2.6 action

回合行动类。该类的成员变量包括：

- **actionType: actions** 行动类型
- **target: int** 行动对象，当 actionType.value 为 0 或 1 时表示目标玩家索引，否则表示目标手牌索引
- **color: colors** 提示颜色，仅当 actionType.value 为 0 时有值
- **number: int** 提示数字，仅当 actionType.value 为 1 时有值

成员方法包括：

- **___init___(actionType, target, **kwargs)**

  构造方法，参数释义：

  (a) actionType: actions 行动类型

  (b) target: int 行动对象

  (c) color=c: colors 关键词参数，提示颜色，仅当 actionType.value 为 0 时需要

  (d) number=n: int 关键词参数，提示数字，仅当 actionType.value 为 1 时需要

2.2.7 game

游戏类。该类的成员变量包括：

- **deck: list** 卡组
- **discard: list** 弃牌堆
- **field: list** 场上的卡牌堆
- **maxScore: int** 能获得的最高分
- **players: list** 玩家
- **hints: int** 提示指示物数量
- **error: int** 错误指示物数量
- **hands: list** 玩家手牌
- **score: int** 当前分数

成员方法包括：

- **___init___(**kwargs)**

  构造方法，参数释义：

  (a) players=ps: list 关键词参数，玩家列表，其中的元素必须继承 basePlayer 类

- **resolveTurn(index, player)**

  接收玩家行动并结算玩家回合，参数释义：

  (a) index: int 当前玩家索引

  (b) player: player 当前玩家对象

- **openingHands()**

  分发起始手牌

- **checkEndgame(index)**

  检查回合是否结束，如结束返回分数，否则返回-1。参数释义：

  (a) index: int 当前玩家索引

- **start()**

  开始该场游戏，结束时返回得分

2.2.8 basePlayer(ABC)

玩家抽象类。成员变量包括：

- **onStart: list** 游戏开始监听器列表
- **onGameEnd: list** 游戏结束监听器列表
- **onTurnStart: list** 回合开始监听器列表
- **onHintColor: list** 收到颜色提示监听器列表
- **onHintNumber: list** 收到数字提示监听器列表
- **onPlayerAction: list** 收到玩家行动监听器列表

抽象方法包括：

- **turn()**

进行一个回合

2.3 游戏周期及监听器

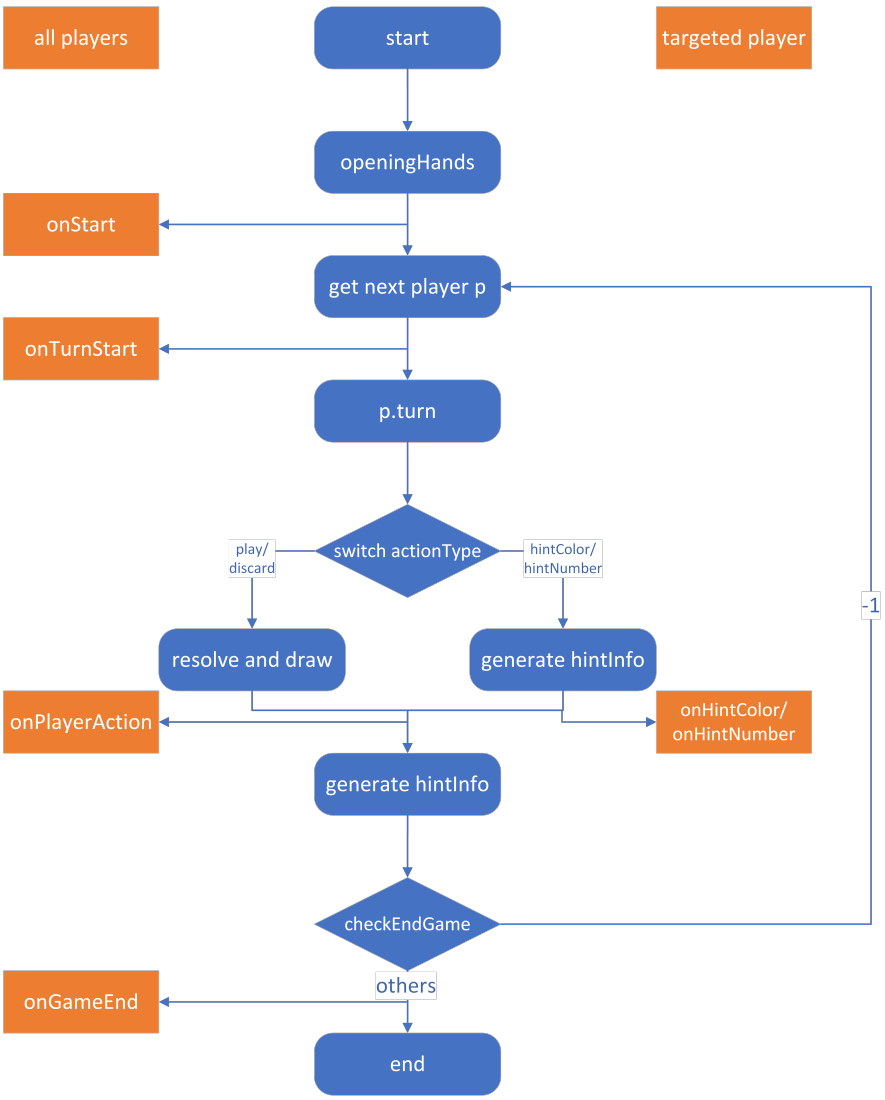
2.3.1 开始一场游戏

使用玩家列表创建一个 game 类的实例后，只需执行其 start 方法即可完整地进行一场游戏。下为从创建玩家到显示一场游戏最终得分的 Python 程序源代码样例:

```
from game import game
from player import player

players = []
for i in range(4):
    players.append(player())
theGame = game(players)
score = theGame.start()
print('Score:{}'.format(score))
```

执行 start 方法后的游戏周期及触发的监听器如下图所示。



2.3.2 监听器

在 basePlayer 类中规定了若干监听器列表，触发时机已在上节中阐述。下为游戏类触发各监听器列表中的方法的形式：

- **onStart: fun(handStatus)**

  handStatus: list 所有玩家的手牌信息，handStatus[i][j] 返回一个 card 对象，表示玩家 i 第 j 张手牌。玩家自己的手牌总是由 card(colors.unknown, -1) 组成。

- **onGameEnd: fun(reason, score)**

  reason: endgames 游戏结束原因

  score: int 终局计分

- **onTurnStart: fun(index)**

  index: int 开始回合的玩家索引

- **onHintColor: fun(hintInfo)**(仅触发目标玩家的该监听器列表)

  hintInfo: list 提示信息，hintInfo[i] 返回一个 card 对象，hintInfo[i].number 必定为-1，hintInfo[i].color 为本次对该玩家第 i 张卡牌的提示信息，若该卡被指出则该值为提示颜色，否则为 colors.unknown。

- **onHintNumber: fun(hintInfo)**(仅触发目标玩家的该监听器列表)

  hintInfo: list 提示信息，hintInfo[i] 返回一个 card 对象，hintInfo[i].color 必定为 colors.unknown，hintInfo[i].number 为本次对该玩家第 i 张卡牌的提示信息，若该卡被指出则该值为提示数字，否则为-1。

- **onPlayerAction: fun(index, playerAction, **kwargs)**

  index: int 进行该动作的玩家

  playerAction: action 玩家进行的动作

  play=p: card 关键词参数，玩家打出的卡牌，仅当 playerAction.actionType.value 为 3 时传入该参数

  discard=d: card 关键词参数，玩家弃置的卡牌，仅当 playerAction.actionType.value 为 2 时传入该参数

  draw=r: card 关键词参数，玩家抽取的卡牌，仅当 playerAction.actionType.value 为 2 或 3 时传入。进行回合的玩家将收到 d.color=colors.unknown,d.number=-1

2.4 玩家类编写要求及说明

1. 类名**必须**为 player

2. **必须**继承 basePlayer 类

   成员变量包含 basePlayer 类规定的所有监听器列表，并重载 turn 方法，同步地而非异步地返回一个 action 对象，包含该回合玩家行动的完整信息。

3. 开发语言不限于 Python

   remotePlayer.py 展示了如何将触发监听器转化为通过 websockets 发送一个 JSON。player 类的构造函数**可以**需要 self 以外的参数，如通信需要的 socket、发送请求的地址等，一个 player 对象也**可以**无法独立收发数据 (见 simpleServer.py)，但通信对象地址**限于**本地 (localhost)，且**必须**提供让一个 player 实例完整进行一场游戏的程序源代码。

4. **不可**在游戏过程中依赖动态的人为输入

   包括但不限于控制台输入、鼠标键盘输入等。换言之，执行用仅包含完成初始化 (该步骤可以包括在本地运行其他程序) 的该 player 的实例的玩家列表初始化 game 类实例的 start 方法所需的运行时间在任何情况下都应是有限的。

5. **不可**修改 game.py

   开发过程中可自由调试，但提交的程序需要能够在使用未修改的 game.py 的情况下完整运行至游戏结束。

2.5 远程玩家类使用说明

2.5.1 概述

在 remotePlayer.py 中实现了一个依赖于 websockets 通信的玩家类，其主要功能是将游戏类触发监听器时传入的参数和监听器类型封装至一个 JSON 对象并将该对象转为字符串发送。其 turn 方法将阻塞进程直至收到回合行动 JSON 对象字符串，并将其转为一个 action 对象。

2.5.2 JSON 对象说明

各监听器触发时转发的 JSON 对象如下所示：

- **onStart**

```
load = {
    event: "onStart",
    handColor: handColor,
    handNumber: handNumber
}
```

其中：

handColor: object　handColor[i][j]=handStatus[i][j].color.value

handNumber: object　handNumber[i][j]=handStatus[i][j].number

- **onGameEnd**

```
load = {
    event: "onGameEnd",
    reason: reason,
    score: score
}
```

其中：

reason: Number　即 reason.value

score: Number

- **onTurnStart**

```
load = {
    event: "onTurnStart",
    index: index
}
```

其中：

index: Number

- **onHintColor**

```
load = {
    event: "onHintColor",
    hintColor: hintColor,
    hintNumber: hintNumber
}
```

其中：

hintColor: object　hintColor[i]=hintInfo[i].color

score: object　hintNumber[i]=hintInfo[i].number

- **onHintNumber**

```
load = {
    event: "onHintNumber",
    hintColor: hintColor,
    hintNumber: hintNumber
}
```

其中：

hintColor: object　hintColor[i]=hintInfo[i].color

score: object　hintNumber[i]=hintInfo[i].number

- **onPlayerAction**

```
load = {
    event: "onPlayerAction",
    actionType: actionType,
    target: target,
    color: color,
    number: number,
    play: play,
```

```
            discard: discard,
            draw: draw
        }
```

其中：

actionType: Number 即 playerAction.actionType.value

target: Number 即 playerAction.target

color: Number 即 playerAction.color.value，仅当 actionType 为 0 时存在该键值对

number: Number 即 playerAction.number，仅当 actionType 为 1 时存在该键值对

play: object 即 [kwargs['play'].color.value, kwargs['play'].number]，仅当 actionType 为 3 时存在该键值对

discard: object 即 [kwargs['discard'].color.value, kwargs['discard'].number]，仅当 actionType 为 2 时存在该键值对

draw: object 即 [kwargs['draw'].color.value, kwargs['draw'].number]，仅当 actionType 为 2 或 3 时存在该键值对

此外 turn 方法发送下述 JSON 对象的字符串：

```
load = {
    event: "turn"
}
```

并需要接收下述 JSON 对象的字符串：

```
load = {
    actionType: actionType,
    target: target,
    color: color,
    number: number,
}
```

其中：

actionType: Number 对应 actions 类规定的动作类型的整型

target: Number

color: Number 仅当 actionType 为 0 时需要有值，对应 colors 类规定的颜色类型的整型

number: Number 仅当 actionType 为 1 时需要有值

2.6 simpleServer.py 与 remoteReceiverPlayer 使用说明

2.6.1 概述

simpleServer 的主要功能为等待指定数量的 websockets 连接，随后以这些 websockets 初始化 remotePlayer.player 类实例，以该玩家列表初始化 game 类实例并进行一场游戏。remoteReceiverPlayer 是一个 Vue3 项目，主要功能为接收 remotePlayer.player 对象的转发，并将结果显示在 web 页面上，也通过 web 页面接收人类玩家的操作并回复。

2.6.2 启动 simpleServer

需已安装 python3 及 pip。在所在目录运行命令行，依次执行：

```
>>pip install -r requirements.txt
>>python simpleServer.py
```

2.6.3 部署 remoteReceiverPlayer

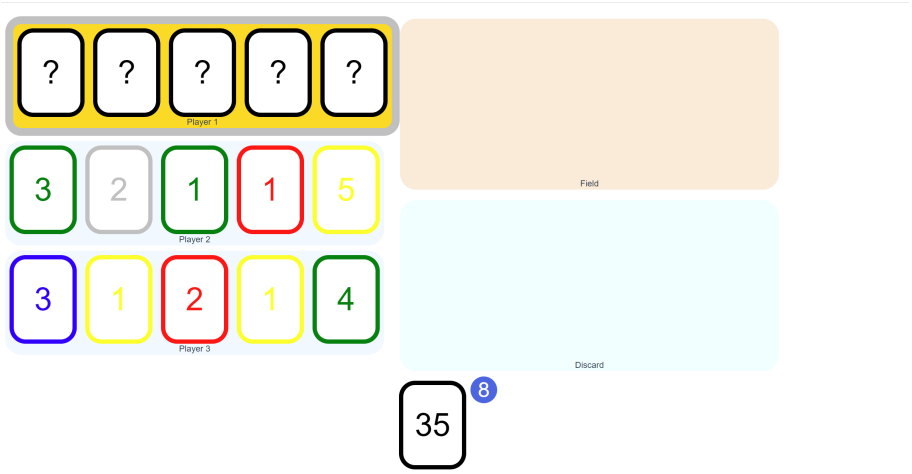
部署该项目需要已安装 npm。在项目目录运行命令行，依次执行：

```
>>npm install
>>npm run dev
```

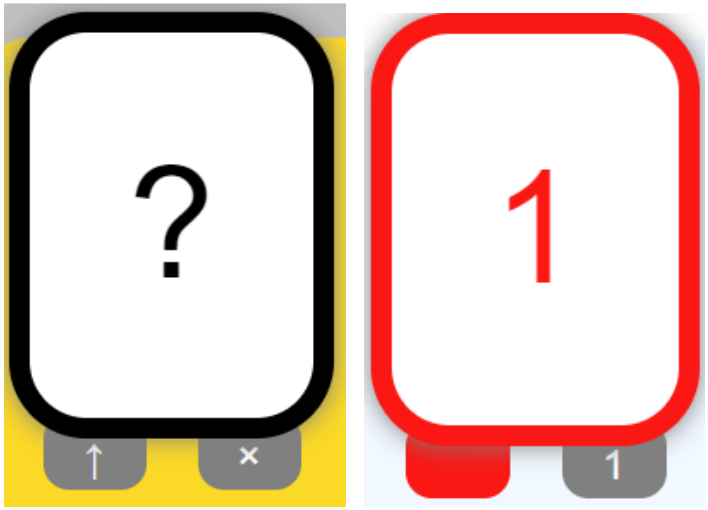
即可通过浏览器访问 (通常地址及端口为 localhost:3000)

2.6.4 界面及操作简介

当玩家数量未达到指定数量时，界面显示 Waiting for players。达到指定数量后如下图所示：

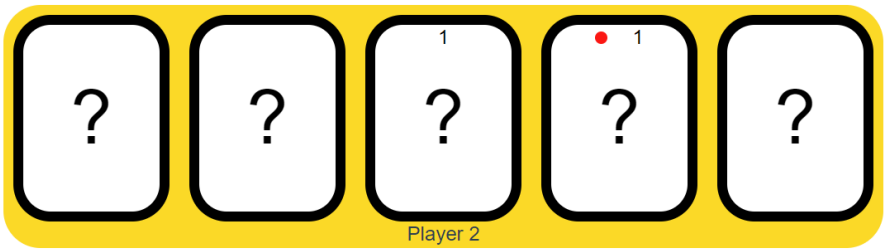


图中左侧为玩家手牌，手牌栏为金色背景表示当前页面玩家，手牌栏有灰色边框表示当前为该玩家回合。右侧从上至下依次为已打出区 (场上)、弃牌区及信息显示区，信息显示区从左至右依次为卡组剩余数量、提示指示物剩余数量及错误指示物数量。正进行回合的玩家单击玩家手牌中的卡牌后根据其是否为自己的手牌分别出现如下图的按钮：

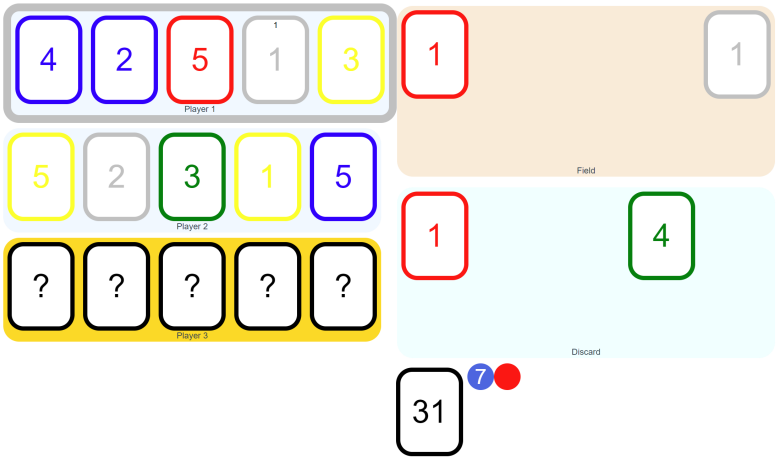


(a) 自己的手牌 (b) 其他玩家手牌

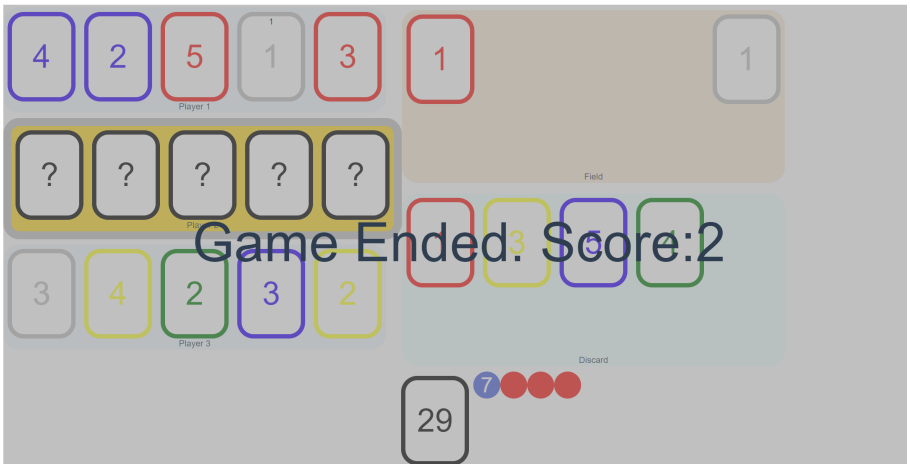
对于自己的手牌，左侧按钮为打出，右侧按钮为弃置。对于其他玩家的手牌，左侧为提示颜色，右侧为提示数字。玩家收到提示后将显示在手牌上方，如下图：



打出和弃置的手牌将分别进入对应的卡堆，如下图所示：



游戏结束时将显示 Game Ended 及分数。如下图：


4
2
5
1
3
Player 1
1
1
Field
? ? ? ? ?
Game Ended: Score:2
3
4
2
3
2
Player 3
Discard
7
29

3 提交要求

3.1 需要额外提交的文件及文件结构

任务步骤 1 至步骤 3 中实现的“玩家类”均需要提交，对于每个步骤分别建立文件夹。每个步骤的提交文件除含玩家类定义的源代码外，需要满足引入 game.py 后能以一或多个文件为入口，有限时间内进行一场包含 4 个该玩家类实例的玩家列表初始化的游戏。此外对于每一个独立运行的部分，须有文本文档记述该部分的库依赖。文件结构如下所示：

```
/
├── SelfCooperation
│   ├── *.py
│   ├── requirements_python.txt
│   ├── *.*
│   └── requirements_*.txt
├── ComprehensiveCooperation
│   ├── *.py
│   ├── requirements_python.txt
│   ├── *.*
│   └── requirements_*.txt
└── RainbowCooperation
    ├── *.py
    ├── requirements_python.txt
    ├── *.*
    └── requirements_*.txt
```

3.2 说明文档

说明文档中需要包含每一个任务步骤提交文件的

1. 玩家类的开发语言及库依赖 (见上节)
2. 如何将你们的电子玩家引入一局游戏中 (参考 2.3.1 节或 2.6 节)
3. 该电子玩家在单场及连续进行的游戏中的行为特征