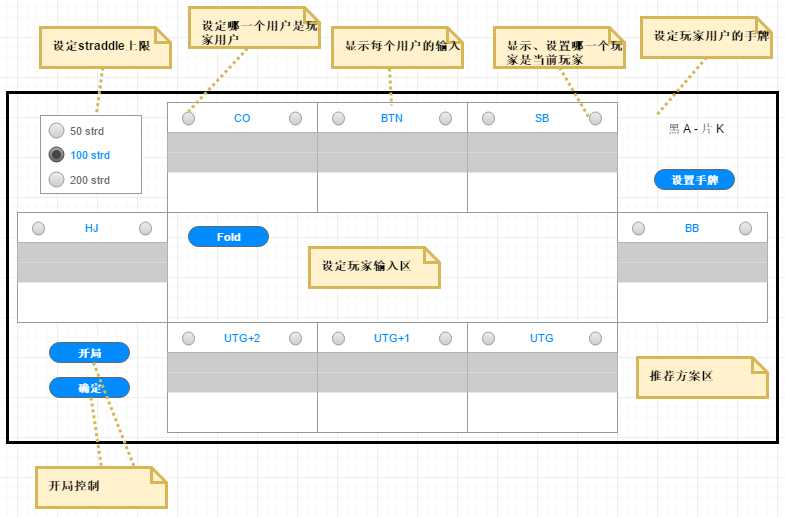
Texas\_Strategy程序设计说明书

# Change Log

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author/modifier | Note |
| 0.1 |  | Tao Zhang | Draft |
| 0.2 |  | Tao Zhang | Draft |
| 0.3 | 2018/09/09 | Tao Zhang | Remove "set\_to\_current" buttons |
| 0.4 | 2018/09/25 | Tao Zhang | Add Controller |
| 0.5 | 2018/09/26 | Tao Zhang | Add Module |
| 0.6 | 2018/09/27 | Tao Zhang | Change the interface format between the GUI and the Controller from json to object. These two parts communicate by calling each other’s methods in Python language level where the json format won’t help.  The analysis program will dump its results into json files which will be used by the TexasStrategy program.  The old GUI program with json process is backed-up as V0.5. |
| 0.7 | 2018/10/04 | Tao Zhang | Change the input logic of hole cards from 4 drop-down menus (option menu) to a dialog with all pairs spreading as buttons. |
| 0.8 | 2018/10/05 | Tao Zhang | If a player has folded already, the program should fold for him/her in later turns. |
| 0.9 | 2018/10/18 | Tao Zhang | 1. Change the unified process mode to the straddle specified process mode; 2. Change the UTG, UTG+1, UTG+2 players to the STR, UTG, UTG+1 players |

# 界面



【V0.9改动（玩家名变更）：

界面中显示的玩家名、界面采集到并传给Controller的玩家名，都是通过一个List常亮在类GUI\_PLAYER\_GROUP中\_\_init\_\_()中定义的。在GUI程序的其他地方没有写死玩家名。但注意，在某些注释中提到了玩家名，需要一并改过来。

Controller程序起到“从GUI抓取信息，传递给strategy程序；然后再以strategy程序的返回，修改GUI显示”的作用，因此其内部不关心玩家名（只是传递）。

Strategy程序，是通过Controller传递过来的GUI抓取的玩家名来进行文件名的匹配，因此只要GUI中的玩家名与文件中的玩家名匹配即可，strategy程序也不介意玩家名。

综上分析，我们需要：

1）将】

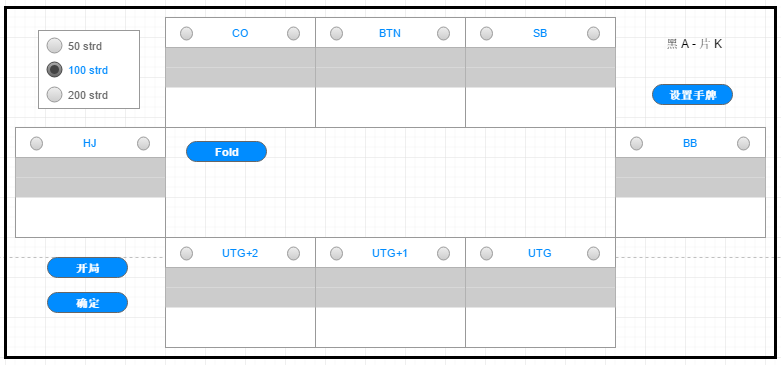
# 事件（GUI）

## 事件列表

### 开局按钮点击

**事件触发动作：**

进入程序后默认状态；



**事件准入检查：**

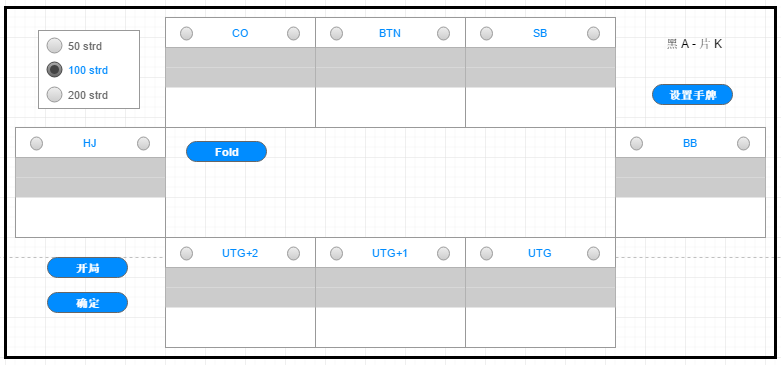
无

**事件响应：**

1. STRD区：设置为缺省strd并激活；
2. 玩家区：
   1. 玩家输入：清空；
   2. 用户位置标示：清空并激活；
   3. 当前玩家标示：清空并置灰；
3. 手牌区：激活“设置手牌”按钮；
4. 动作区：所有动作清空并置灰【最多的有30个输入，要设置30个按钮】；
5. 开局区：
   1. “开始”按钮激活；
   2. “确定”按钮激活；
6. 推荐区：清除

### “设置手牌”按钮点击

**事件触发动作：**

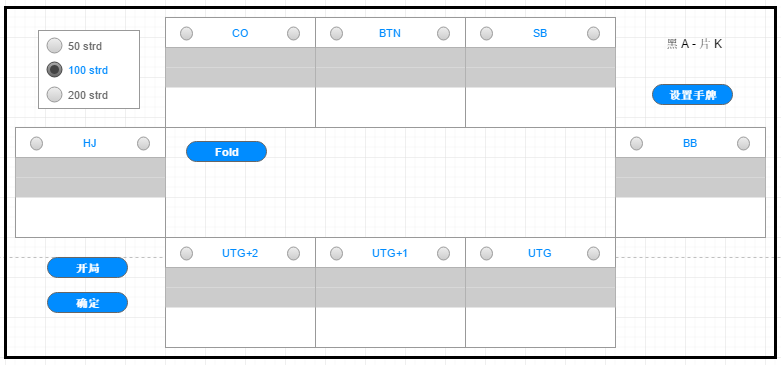


**事件准入检查：**无

**事件响应：**

### 开局确定按钮点击

**事件触发动作：**

****

**事件准入检查：**

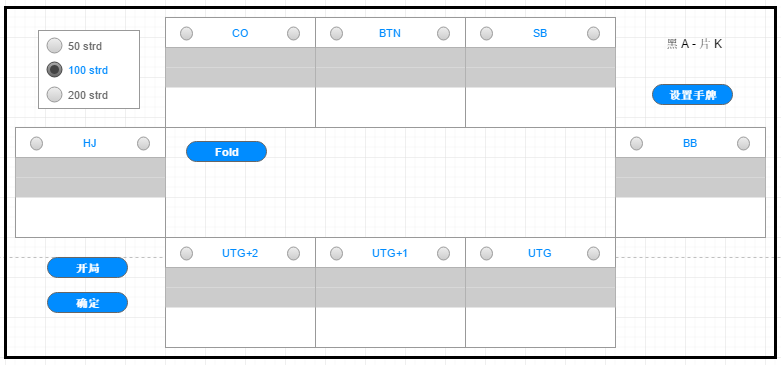
1. STRD区：是否设置
   1. 设置：无动作；
   2. 没有设置：弹出警告对话框“尚未设置本局straddle限制”，取消事件响应；
2. 玩家区：是否设置用户位置标示；
   1. 设置：无动作；
   2. 没有设置：弹出警告对话框“尚未设置用户位置”，取消事件响应；
3. 手牌区：是否设置了两张手牌；
   1. 两张手牌都设置了：无动作；
   2. 未设置两张手牌：弹出警告对话框“请设置手牌”，取消事件响应；
4. 动作区：无；
5. 开局区：无；
6. 推荐区：无

**事件响应：**

1. STRD区：置灰；
2. 玩家区：
   1. 玩家输入：设置SB、BB、UTG的输入【考虑引入机器人的输入机制以统一所有动作，即考虑在点击“开局确定”按钮后，并不直接设置这三玩家的输入，而是进入等待SB动作的状态，但是通过机器人输入SB的动作；然后同理自动进行BB、UTG的动作，这样所有的处理能够在一定程度上得到统一，在以后规则变化的情况下，也好扩展、修改】【考虑到SB、BB、UTG的首次输入并不纳入输入的计算体系，所以可能还是不能这样处理】【干脆以UTG+1作为第一个玩家，不显示SB\BB\UTG的输入就好了】；
   2. 用户位置标示：置灰；
   3. 当前玩家标示：设置到UTG+1（或SB）（见上面的注），激活；
3. 手牌区：将“设置手牌”按钮置灰；
4. 动作区：显示当前玩家的可能动作，激活相应按钮，取消其他按钮显示并置灰（见上面的注）；
5. 开局区：
   1. “开始”按钮：无动作（持续激活）；
   2. “确定”按钮：置灰；
6. 推荐区：判断是否当前玩家==用户位置
   1. 是：显示推荐策略；
   2. 否：清空显示

### 动作按钮点击

**事件触发动作：**



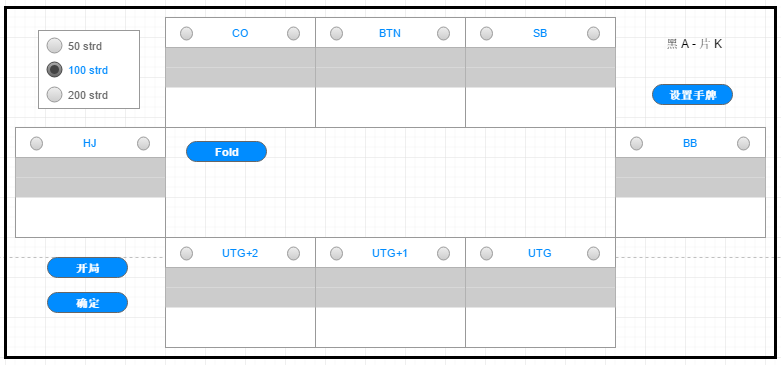
**事件准入检查：**无

**事件响应：**

1. STRD区：无动作（持续灰着）；
2. 玩家区：
   1. 玩家输入：判断当前输入是否为最新输入（最新输入表示本次输入为正常的顺序输入，非最新输入表示本次输入是因点击了“设为当前”按钮过来的修正补录）
      1. 是最新输入：将选择的动作显示到对应玩家（最新）输入上；
      2. 非最新输入：将对应玩家的最后一次输入替换为选择的动作；
   2. 用户位置标示：无动作（持续灰着）；
   3. 当前玩家标示：设置到下一玩家（持续激活）；
3. 手牌区：无动作（持续灰着）；
4. 动作区：显示下一玩家的可能动作，激活相应按钮，取消其他按钮显示并置灰；
5. 开局区：
   1. “开始”按钮：无动作（持续激活）；
   2. “确定”按钮：无动作（持续置灰）；
6. 推荐区：判断是否下一玩家==用户位置
   1. 是：显示推荐策略；
   2. 否：清空显示

### “当前玩家”按钮点击

**事件触发动作：**

****

**事件准入检查：**

1. 玩家区：
   1. ~~选中玩家是否为当前玩家：~~【不必进行此判断，选中当前玩家，就是和没动作一样】
      1. ~~是；当前玩家是否已经有过动作：~~
         1. ~~是：弹出警告对话框“不能修改上轮动作，请为用户选择本轮动作”，取消事件响应；~~
         2. ~~否：取消事件响应；~~
      2. ~~否；~~当前玩家是否已经有过动作：
         1. 是：无动作（通过检查）；
         2. 否：弹出警告对话框“本用户尚未进行动作”，将当前玩家标识置回，取消事件响应；

**事件响应：**

1. STRD区：无动作（持续灰着）；
2. 玩家区：
   1. 玩家输入：无动作；
   2. 用户位置标示：无动作（持续灰着）；
   3. 当前玩家标示：设置到该玩家（持续激活）；
3. 手牌区：无动作（持续灰着）；
4. 动作区：显示该玩家的可能动作，激活相应按钮，取消其他按钮显示并置灰；
5. 开局区：
   1. “开始”按钮：无动作（持续激活）；
   2. “确定”按钮：置灰；
6. 推荐区：判断是否设定玩家==用户位置
   1. 是：显示推荐策略；
   2. 否：清空显示

## 状态列表

### 开局状态 (open\_state)

状态目的：等待用户设置开局条件，包括：straddle限制、手牌、用户位置。

**各区域情况：**

1. STRD区：激活状态；
2. 玩家区：
   1. 用户位置：激活状态；
   2. 当前玩家：非激活状态；
   3. 玩家动作：
3. 手牌区：激活状态；
4. 动作区：非激活状态；
5. 开局区：
   1. “开始”按钮：激活状态；
   2. “确定”按钮：激活状态；
6. 推荐区：

### 牌局进行状态 (running\_state)

状态目的：等待用户输入牌局进行信息。

**各区域情况：**

1. STRD区：非激活状态；
2. 玩家区：
   1. 用户位置：非激活状态；
   2. 当前玩家：激活状态；
   3. 玩家动作：
3. 手牌区：非激活状态；
4. 动作区：
   1. 当前可用按钮：激活状态；
   2. 当前不可用按钮：非激活状态；
5. 开局区：
   1. “开始”按钮：激活状态；
   2. “确定”按钮：非激活状态；
6. 推荐区：

## 状态转移流程图

开局

按钮点击

开局状态

牌局进行状态

开局确定

按钮点击

动作

按钮点击

当前玩家

按钮点击

# 对象设计（GUI）

## STRD区

### set\_open\_state()

设置为缺省strd并激活strd（radio button）

### get\_strd\_limit()

返回当前选中的strddle limitation（int）

### set\_running\_state ()

将strd（radio button）置灰

## 手牌区

### set\_open\_state()

设置为“空-空”+“空-空”并激活

### get\_hole\_cards()

返回当前设置的hole cards（([s1, v1], [s2, v2])）

### set\_running\_state ()

将hole cards区置灰

## 玩家区（整体）

### get\_user\_pos()

返回用户位置（var\_user\_pos，控制变量），如果未设置，则返回0

### get\_user\_name()

返回用户名称

### get\_current\_pos()

返回当前玩家位置（var\_current\_pos，控制变量），如果未设置，则返回0

### get\_current\_player\_name()

返回当前玩家名称

### get\_latest\_next\_pos()

返回最后动作玩家的下一个玩家的位置（latest\_next\_pos）

举例：开局时，本方法返回UTG+1位置

注：点击“当前玩家”按钮并不会影响最后动作玩家的位置

### next\_latest\_next\_pos ()

将latest\_next\_pos设置到下一个位置（从0～7循环）

### next\_current\_pos ()

将var\_current\_pos设置到下一个位置（从0～7循环）

### set\_open\_state()

清空所有玩家动作【调用所有单体方法clear\_actions()】；

清空用户位置标示var\_user\_pos；

激活用户位置标示【调用所有单体方法enable\_user\_pos()】

清空当前玩家标示var\_current\_pos；

将当前玩家标示置灰【调用所有单体方法disable\_current\_pos()】

清空latest\_next\_pos(-1)

### set\_running\_state()

增加SB、BB、UTG的动作【调用三者单体方法set\_action()】；

置灰用户位置标示【调用所有单体方法disable\_user\_pos()】；

保存用户位置到相关var\_user\_pos（控制变量，不用保存自己会变）；

将var\_current\_pos设置为UTG+1（显示，控制变量）；

激活当前用户标识按钮【调用所有单体方法enable\_current\_pos()】

将latest\_next\_pos设置到UTG+1（不显示，非控制变量）；

### process\_player\_action(action)

进行设置前，需要进行如下检查：

* 判断var\_current\_pos是否设置；若无，则报错“尚无当前玩家”（按理说无法出现这种情况）；
* 判断var\_user\_pos是否设置；若无，则报错“未设置用户位置” （按理说无法出现这种情况）；
* 判断latest\_next\_pos是否设置；若无，则报错“未设置用户位置” （按理说无法出现这种情况）；

判断：

* 如果latest\_next\_pos与var\_current\_pos的指向同一个位置（说明是为当前玩家增加新的动作）：
  + 将输入动作增加到当前玩家的动作【调用var\_current\_pos对应的单体的方法set\_action(action, True)】
  + latest\_next\_pos步进（调用next\_latest\_next\_pos ()）
  + var\_current\_pos步进（调用next\_current\_pos()）
* 如果latest\_next\_pos与var\_current\_pos的指向不同位置（说明是为当前玩家修改最近动作）：
  + 用输入动作替换当前玩家的最后一个动作【调用var\_current\_pos对应的单体的方法set\_action(action, False)】
  + var\_current\_pos步进（调用next\_current\_pos()）

### get\_players\_all\_actions()

~~返回json players\_actions（之所以选择json，是为了前后台分层，即如果有一天前后台某一部分不在使用python，那么通过json这种通用的信息标准来传递信息更加通用与标准）~~

~~该json串是由如下结构构成：~~返回如下对象

List [ # 玩家是有顺序的，因此使用list

Dict { # 一个玩家对应多个动作，因此使用dict

‘player\_name’: list [actions] # 某一玩家的动作是有序的，因此使用list

},

Dict …

]

通过self.player\_name映射出每个玩家的名字；

通过调用各玩家的get\_action得到其action（list）

### get\_players\_valid\_actions()

get\_players\_all\_actions包含所有玩家的所有动作，但假如这时用户将current\_player设置（reset）到了用户玩家身上，那么我们这时候使用所有动作来寻找推荐策略是没有意义的。

我们可以：1）丢弃“用户玩家前一用户的最后一轮输入”之后玩家的同轮输入；2）类似前者，但是不是以用户玩家为基准，而是以当前玩家为基准。

选择方案2，因为方案2的解决空间是方案1的超集

## 玩家区（单体）

### clear\_actions()

清空该玩家所有动作（显示与动作列表一致，所以SB, BB, UTG等输入都在其中）：

* 将action\_list清空（action\_list是一个list，其中item为控制动作显示label的var\_string，清空意味着将var\_string置为空字符串）；

### set\_action(action, is\_new)

设置玩家动作。如果is\_new=Ture，则在设置在action\_list中最开始一个空的var上；如果is\_new=False，则设置在action\_list中最后一个非空的var上

### get\_actions()

返回变长list actions，其item为字符串，顺序对应本玩家所有动作

### get\_previous\_action()

返回本玩家上一个动作，如果没有，则返回空字符串。

### enable\_user\_pos()

激活用户位置标示

### disable\_user\_pos()

置灰用户位置标示

### enable\_current\_pos()

激活当前玩家标示

### disable\_current\_pos()

置灰当前玩家标示

## 开局区

### \_\_init\_\_()

创建“开局”按钮，

创建“确定”按钮，置灰

### set\_open\_state()

将“确定”按钮激活（“开局”按钮从\_\_init\_\_起一直处于激活状态，没必要处理）

### set\_running\_state()

将“确定”按钮置灰

## 推荐区

### \_\_init\_\_()

推荐区最多有4行label，用于显示对用户的推荐；初始化为空。

### set\_advice(advice)

输入advice为一个list，每一个item为一个string，advice最多包括4个items；此将advice labels置为对应item中的string

## 动作区

### \_\_init\_\_()

初始化30按钮（5行6列），所有按钮存放在list action\_button\_list中，通过action\_button\_list[action\_idx]来访问每一个按钮。

每按钮的显示由一个textvariable控制，所有textvariable存放在list action\_list中，通过action\_list[action\_idx]来访问每一个按钮的

注：按照如下方法来进行处理绑定：

e.g.

>>> fs=[]

>>> for i in range(4):

def lam(x=i):

return x\*x

fs.append(lam)

>>> fs

[<function lam at 0x000000000473EF28>, <function lam at 0x0000000004732488>, <function lam at 0x00000000047322F0>, <function lam at 0x0000000004732510>]

>>> fs[1]()

1

>>> fs[2]()

4

>>>

### set\_open\_state()

清空所有动作按钮的显示，置灰所有动作按钮

### set\_possible\_actions(possible\_action\_list)

先调用set\_open\_state()，以清空所有按钮的显示并置灰；

possible\_action\_list中每一个item是一个str，从0开始将action\_list中的每一个var设置为possible\_action\_list中的item（str）

# 与GUI交互的Controller方法

## 事件-开局按钮点击（open\_event\_process）

**检查：无**

**设置：**

STRD区：strd.set\_open\_state()

玩家区：player\_group.set\_open\_state()

手牌区：hole\_cards.set\_open\_state()

动作区：actions.set\_open\_state()

开局区：start.set\_open\_state()

推荐区：advice.set\_advice(空)

## 事件-开局确定按钮点击（run\_event\_process）

**检查：**

if strd.get\_strd\_limit() == 0: warning("Please set straddle limitation first") & exit

if player\_group.get\_user\_pos() == 0: warning("Please set user position first") & exit

手牌区，见前

**设置：**

STRD区：strd.set\_running\_state()

玩家区：player\_group.set\_running\_state()

手牌区：hole\_cards.set\_running\_state()

动作区：actions.set\_possible\_action('some actions') --- 需要后台支持

开局区：start.set\_running\_state()

推荐区：

if player\_group.get\_user\_pos() == player\_group.get\_current\_pos():

advice.set\_advice('some advice') --- 需要后台支持

else:

advice.set\_advice(空)

## 事件-动作按钮点击（act\_event\_process）

**检查：无**

**设置&动作：**

不同action按钮，将其索引传递给了按动触发的函数（见动作区\_\_init\_\_()的相关例子）

Controller保存当前动作列表，通过该列表与索引，我们可以得到相关动作action

Play\_group.process\_player\_action(action)

actions.set\_possible\_action('some actions') --- 需要后台支持

推荐区：

if player\_group.get\_user\_pos() == player\_group.get\_current\_pos():

advice.set\_advice('some advice') --- 需要后台支持

else:

advice.set\_advice(空)

【V0.8：在刘建威新需求中，要求当一个玩家已经Fold的时候，轮转到他/她的时候，程序自动为其进行Fold，而无需用户进行输入。为此我们检查了程序，确认动作是由“不同action按钮，将其索引传递给了按动触发的函数”即：执行GUI\_ACTION.\_\_init\_\_()方法中的act\_event\_process(act\_num)。那么要想自动执行Fold，我们需要在合适的地方执行这个act\_event\_process(act\_num)。先看act\_num，因为Fold是第0个动作，所以num为0。再看act\_event\_process，这个方法是作为初始化参数，由类GUI传递给GUI\_ACTION的；而类GUI是从Controller那里得到的（Controller对象作为GUI的一部分，所以GUI能引用Controller的方法），而且更好的一点是，一直到Controller，都只有act\_num一个参数。考虑到运算逻辑都通过Controller或Module实现，所以我们在Controller进行判断，看看当前玩家的上一个动作是否是Fold，如果是，我们在act\_event\_process中自己调用自己（使用参数0）即可实现~~【注：这里不能完全的自己调用自己，否则，将会陷入一个死循环中。改变的方式：将自动调用和实际触发分开，自动调用时不再进行检查，这样自动调用就能够返回而不触发递归死循环了。为此，对Controller的act\_event\_process引入一个参数：auto\_process=[~~**~~False~~**~~|True]】。~~【前面这段有问题，我们可以自己调用自己，只不过是在处理完当前动作之后，因为本身这个动作是手工触发的，所以自动触发不应该处理本动作，而应该处理下一个动作】为此，还需要修改GUI程序，使Controller能获取当前玩家的上一个动作。

GUI程序：玩家区整体类，增加方法（返回当前玩家的上一个动作）：get\_current\_player\_previous\_action；玩家区单体类，增加方法（返回本玩家上一个动作）：get\_previous\_action】

## 事件-“当前玩家”按钮点击（set\_current\_event\_process）

**检查：**

当前玩家是否已经有过动作（此检查在GUI部分完成，因为主要是player自己的检查以及player\_group的设置，不涉及其他控件或后台逻辑）

**设置&动作：**

只动作区与推荐区有动作（与动作按钮点击事件类似）

actions.set\_possible\_action('some actions') --- 需要后台支持

推荐区：

if player\_group.get\_user\_pos() == player\_group.get\_current\_pos():

advice.set\_advice('some advice') --- 需要后台支持

else:

advice.set\_advice(空)

# 与Module交互的Controller方法

## set\_act (view)

Controlller从GUI获取:

* 当前straddle限制（strd.get\_strd\_limit），
* 用户名（play\_group.get\_user\_name），
* player名（player\_group.get\_current\_player\_name）

传递给module.get\_possible\_actions，

得到possible\_action\_list，将该list返回GUI

## set\_advice(view)

Controlller从GUI获取:

* 当前straddle限制（strd.get\_strd\_limit），
* 用户名（play\_group.get\_user\_name），
* 当前valid\_action\_group（get\_players\_valid\_actions）
* 用户手牌（hole\_cards.get\_hole\_cards）
* ~~与player\_pos(get\_current\_pos)【注：pos是从1开始，idx是从0开始，有空的时候梳理一下】~~

传递给module.get\_advice，得到advice\_list，将该list返回GUI

# Module方法

## \_\_init\_\_ ()

从上面的分析中可见，straddle与user\_name需要在整个对象中留存一份，即需要定义self.current\_strad、self.current\_user\_name，其初始值为0与""。

从后面的get\_advice设计中可见，需要配置文件提供根目录（strategy\_files\_root），因此\_\_init\_\_()中需要读取配置文件（使用configparser）。

下面的代码是将所有hole\_cards的可能组合（1326种）在strategy file中的位置设置好，以便后面通过hole cards来找推荐打法在strategy file中的位置【最开始的时候，此部分代码是放到read\_strategy\_file方法中，后来想到此组合在任何一次牌局的任何一次查找推荐打法时都是一样的，所以我们在\_\_init\_\_()执行一次即可，在read\_strategy\_file中，我们使用此组合即可】

这里的代码是示例代码：

l = []

for value in ['2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']:

for suit in ['黑', '红', '片', '梅']:

l.append([suit, value])

hc = []

for card1 in l[1:]:

for card2 in l[:l.index(card1)]:

hc.append([card1, card2])

这里的代码是真实代码：

cards = []

for value in ['2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']:

for suit in ['黑', '红', '片', '梅']:

cards.append([suit, value])

self.pairs = []

for card1 in cards[1:]:

for card2 in cards[:cards.index(card1)]:

self.pairs.append([card1, card2])

## get\_possible\_actions(straddle, user\_name, player\_name)

如果传入的straddle限制、user\_name与当前一致（self.current\_strad、self.current\_user\_name），则直接从self.current\_possible\_actions读取对应player的possible\_action；

否则，修改self.current\_strad、self.current\_user\_name，并通过这两者找到存放可能动作的json文件，将该文件读入self.current\_possible\_actions，再读取对应player的possible\_action。

【在文档编写到此处之前，忘记了straddle这一参数；在此识别到后，修改前面的设计，包括Controller的set\_act；梳理PioRes时，发现还忘了user\_name参数，只好继续修改前面的设计，包括Controller的set\_ac，并给GUI.player\_group增加方法get\_user\_name】

## get\_advice (straddle, user\_name, ~~player\_pos,~~ action\_list, hole\_cards)

~~如果传入的straddle限制、user\_name与当前不一致（self.current\_strad、self.current\_user\_name），先进行修改；~~

使用get\_advice\_file\_name (user\_name, action\_list)取得文件名。按照配置文件中的根目录、strddle限制与user\_name找到相应strategy文件目录，然后使用glob之类的方法寻找该文件。

如果未找到该文件，返回advice：[‘No suggestion’, ‘in PioSOLVER’]

如果找到该文件（取得绝对路径文件名），则调用read\_strategy\_file读取出策略并返回。

## get\_advice\_file\_name (user\_name, action\_list)

根据action\_list与user\_name拼出strategy文件名（示例代码见下）：

a = [{'SB':['SB', '14.1strd']}, {'BB':['BB', '32.1strd']}, {'UTG':['2BB', 'FOLD']}, {'UTG+1':['FOLD', 'FOLD']}, {'UTG+2':['FOLD', 'FOLD']}, {'HJ':['FOLD', 'FOLD']}, {'CO':['FOLD', 'FOLD']}, {'BTN':['3.5strd', 'AllIn']}]

s = ''

for i in range(4):

for player in a:

for key in player:

if len(player[key]) > i:

if player[key][i] not in ['SB', 'BB', '2BB', 'FOLD']:

s += key + '\_' + player[key][i] + '\_'

加上后缀就可以返回了

## read\_strategy\_file (absolute\_file\_name, hole\_cards)

根据文件名打开文件

~~l = []~~

~~for value in ['2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']:~~

~~for suit in ['黑', '红', '片', '梅']:~~

~~l.append([suit, value])~~

~~hc = []~~

~~for card1 in l[1:]:~~

~~for card2 in l[:l.index(card1)]:~~

~~hc.append([card1, card2])~~

注：上面这部分代码实际上是为了算出所有hole\_cards的可能组合（1326种）在strategy file中的位置，对于任一牌局来说，这1326种组合及其位置都是固定的，因此我们不应该在每次查找推荐打法的时候都设置一遍，所以我们把这部分代码放到\_\_init\_\_()中了。

为了

# analysePioRes程序的输出

## possible\_actions\_*strd*\_*userName*.json

文件名例子：

possible\_actions\_100strd\_UTG+1.json

possible\_actions\_50strd\_BTN.json

文件内容例子：

{'BB':['Fold', 'Call', '5.3strd', '6.0strd', '14.1strd', '15.8strd', '16.6strd', '17.5strd', '30.0strd', '30.3strd', '31.8strd', '32.1strd'],'UTG':['32.8strd', '22.0strd', '5.3strd', '18.0strd', '17.5strd', '32.1strd', 'Fold', 'Call', '6.0strd', '15.8strd', 'AllIn'],'BTN':['32.0strd', '32.1strd', '14.1strd', 'Fold', '40.4strd', '3.5strd', 'Call', '24.0strd', '10.6strd', 'AllIn'], 'CO':['32.0strd', '32.1strd', '14.1strd', 'Fold', '3.5strd', 'Call', '24.0strd', '10.6strd', 'AllIn'], 'HJ':['32.1strd', '14.1strd', 'Fold', '3.5strd', 'Call', '24.0strd', '10.6strd', 'AllIn'], 'SB':['30.2strd', '32.8strd', '14.1strd', 'Fold', '17.6strd', 'Call', '24.0strd', '30.0strd', '5.5strd', 'AllIn', '31.9strd'],'UTG+1':['32.1strd', 'Fold', '3.5strd', 'Call', 'AllIn'], 'UTG+2': ['32.1strd', 'Fold', '3.5strd', 'Call', '10.6strd', 'AllIn']}

应该按照上面BB的样子进行排序后，再输出到文件中（后面的player未排序）

注意：strd和前面的数字之间不要有空格（因为在后面查找strategy文件名时，数字与strd之间是没有空格的，从头就不加空格，避免了后续所有的麻烦）

# analysePioRes设计

## \_\_init\_\_ ()

程序需要从配置文件中读取PioSOVLER文件所在的目录。

初始化一些全局变量。

最重要的全局变量为:self.strategy\_files：带结构的所有strategy文件名字典（原来命名为strd\_dict，后改了）。读取完成后，self.strategy\_files的内容应该为：{strd\_limit: {user\_position: [file\_names]}

样例：

{'100straddles':

{'BB':

['BB\_Call\_UTG\_5.3strd\_BB\_strategy.txt',

'BB\_strategy.txt',

'BTN\_3.5strd\_BB\_strategy.txt',

'BTN\_3.5strd\_BB\_14.1strd\_UTG\_32.1strd\_BTN\_Fold\_BB\_strategy.txt',

'CO\_3.5strd\_BTN\_Call\_BB\_17.5strd\_CO\_AllIn\_BTN\_Call\_BB\_strategy.txt',

'HJ\_3.5strd\_BB\_Call\_UTG\_17.5strd\_HJ\_AllIn\_BB\_strategy.txt'],

'UTG+2':

['UTG+1\_3.5strd\_UTG+2\_strategy.txt',

'UTG+2\_strategy.txt',

'UTG+1\_3.5strd\_UTG+2\_Call\_BB\_17.5strd\_UTG+1\_Call\_UTG+2\_strategy.txt',

'UTG+2\_3.5strd\_BB\_14.1strd\_UTG+2\_strategy.txt',

'UTG+2\_3.5strd\_CO\_10.6strd\_UTG+2\_strategy.txt',

'UTG+2\_3.5strd\_HJ\_10.6strd\_UTG+2\_strategy.txt']}}

其他全局变量为一些分析结果如：res\_possible\_actions, res\_max\_round, res\_max\_round\_file等

## read\_pio\_result\_files()

读取所有strategy文件，生成self.strategy\_files。

此方法是最核心的处理之一。

## print\_raw\_data(strd, position, num)

本方法是调试使用的，作用是打印某straddle条件下，某玩家position的前num条strategy\_file。

由于strategy\_file是analysis程序中最重要的内部数据，因此检查它正确与否是非常重要的，因此编写了此方法。但：

1）此名字并不好，后续也没想到什么更好的名字，索性就沿用吧；

2）其实还有一种更好的检验方法，即设计一个小的样本集，让analysis程序在此小样本集上进行分析，从而更好的进行验证（我们后来就是用这个办法进行的验证，我们在配置文件中配置了一个测试目录Pio\_Test\_Path，该目录沿用PioSOLVER的结构，但仅仅放了很少的一些文件，从而能很好地进行检验。上面\_\_init\_\_()中的self.strategy\_files实例就是Test目录下的文件）。

## get\_strategy\_files()

将self.strategy\_files返回。

## get\_strddle\_list ()

返回PioSOLVER包含的所有straddle。

【Q：为什么有些result需要先进行analyse，有些result可以直接返回？A：如果需要复杂分析才能得出的result，就需要先进行analyse，然后再通过get方法返回；有些result，比如PioSOLVER中包含的所有straddle，可以直接从strategy\_files中很简单地得到，就不再用analyse方法了】

## analyse\_possible\_actions()

通过对strategy\_files的分析，得出某straddle下某玩家的可能actions。

其结果（样例）为：

{'100straddles':

{'BB':

{'BB': {'14.1strd', '17.5strd', 'Call', 'Fold'},

'UTG':{'32.1strd', '17.5strd', 'Fold', '5.3strd'},

'BTN': {'3.5strd', 'Call', 'Fold'},

'CO': {'AllIn', '3.5strd', 'Fold'},

'HJ': {'AllIn', '3.5strd', 'Fold'}},

'UTG+2':

{'UTG+1': {'3.5strd', 'Call', 'Fold'},

'UTG+2': {'3.5strd', 'Call', 'Fold'},

'BB': {'14.1strd', '17.5strd', 'Fold'},

'CO': {'10.6strd', 'Fold'},

'HJ': {'10.6strd', 'Fold'}}}}

## get\_possible\_actions()

返回analyse\_possible\_actions的结果（self.res\_possible\_actions）。

## save\_possible\_actions(fmt='json')

目前本方法只支持json格式，其他格式目前尚无必要支持。

输出的文件应该是module.get\_possible\_actions(straddle, user\_name, player\_name)中所要求的文件。

# preProcessPioRes

通过之前的单元测试，发现PioSOLVER的strategy文件名存在一个不一致的地方：有些文件名中是没有任何Fold的，但有些文件名中有Fold，对此，我们需要进行预处理使之一致，以便后续的处理。为此，我们需要一个预处理程序：preProcessPioRes。

这个预处理程序可能还会处理以后发现的其他问题；而且在不同版本的PioSOLVER结果集中，可能存在着不同的问题（如，本次结果集存在问题1，问题2，问题3；下次的结果存在问题1，问题3，问题4，问题5）。所以，我们依旧采取类-对象的方式来进行处理（每一个问题对应类的一个方法），而不是简单的过程。

## \_\_init\_\_ ()

程序需要从配置文件中读取PioSOVLER文件所在的目录。

## process\_remove\_fold\_pair()

直接写的代码。。。

# 测试