德州扑克 AI

PPCA2014

July 2014

PPCA2014 的大作业是实现一个德州扑克的 AI。我们会不定期在线进行测试，能够适应不同局势、赢得尽量多的筹码的 AI 会得到更高的分数。本文将介绍本次大作业采用的德州扑克规则 (第1节)，评测系统的使用（第2节），和本次大作业的具体要求（第3节）。

本次大作业的想法来源于贾枭学长，评测系统的框架是他完成的，在此特别致谢贾枭学长。

1 游戏规则

德州扑克使用大小王除外的一副牌，共 52 张。玩家根据手中 2 张底牌和场上 5 张公共牌组合出的最好牌型决定胜负。每一局的流程如下：

1. 洗牌
2. 担任小盲注和大盲注的玩家下盲注（blinds）
3. 庄家（dealer）为每人发两张底牌（hole card）
4. 翻牌前的一轮下注（preflop）
5. 销一张牌，翻三张公共牌（flop）
6. 第二轮下注
7. 销一张牌，翻一张公共牌（turn）
8. 第三轮下注
9. 销一张牌，翻一张公共牌（river）
10. 第四轮下注
11. 若场上还剩至少两名玩家未盖牌，则他们需要展示手牌比较大小（showdown）

12. 分配彩金

一局游戏中至少有 2 名玩家，庄家由玩家轮流担任，沿顺时针方向轮换，小盲注是庄家顺时针方向下一个玩家，大盲注是小盲注顺时针方向的下一个玩家（在 2 人局中就是庄家）。大盲注是小盲注的 2 倍，小盲注将随游戏进行增加，目前每 3 局增加一次, 小盲注的大小依次为 1，2,5,10,20,50,100,200,500，共 27 局。玩家在初始时将获得一样多的筹码，目前为 1000。当一名玩家手中筹码变为 0 时，他就出局了，他只能观察之后的比赛。当场上只剩一名玩家或打满 27 局时，游戏结束。

第一轮下注从大盲注顺时针方向的下一名玩家开始沿顺时针方向进行，大小盲注在此时不算已经下注，但是大小盲注计入第一轮的彩池。后三轮下注都从庄家的下一名玩家开始。当一轮彩池中没有筹码时，玩家可以选择过牌 (check)，盖牌（fold），或加注 (raise)。当有彩池中已有筹码时，玩家可以选择盖牌，跟注（call），或加注。本次大作业采用的是无限下注德州扑克（no-limit），加注时只需要不少于当轮上一个加注的数量，如果玩家是当轮第一个下注的，他的下注至少与大盲注相同。如果一个玩家在想要跟注或加注时，他拥有的筹码不足最低限额，则他依然可以完成通过全押（all-in）进行跟注（超出最高注的部分不算加注，但其他玩家需要跟注到新的最高值）。盖牌的玩家将损失本局游戏中投入的所有筹码，不再参与本局游戏。当以下所有条件满足时一轮下注结束：

- 每个未盖牌的玩家均行动过
- 除了全押的玩家，所有未盖牌的玩家下注都相同
- 下一个行动的玩家恰好是上一个加注的玩家，或者场上只有一名玩家未盖牌，或者所有玩家选择过牌

最终，各个彩池分给该彩池贡献者牌型最大的玩家。若牌型相同，则比较次要牌。依次类推。如果一个彩池的贡献者中有多人牌一样大，则该彩池由他们平分，若彩池中筹码数不能被除尽，则余数被分给该彩池胜者中顺时针方向最靠近庄家的一个。

牌型规则可参考Wikipedia:Texas Hold'em.

目前我们的规则和通常的德州扑克有一些不一致：

- 两人局的盲注依然按照正常局算
- 在第一轮加注中，若彩池中贡献最多的玩家的下注等于大盲注的大小时，大盲注需要跟注 0 个筹码而不是过牌。
- 可能还有其他不一致之处，发现后请指出

2 评测系统

评测系统由 client 和 server 组成，大作业只需要实现 client 中 Player 类的函数即可。评测时，请运行 client，连接指定的服务器进行评测。下面主要介绍 client 的运行环境、编译方法和为实现 AI 提供的接口。

2.1 运行环境和编译方法

评测系统可以在 Linux，Windows，Mac OS 上运行，源代码公开在 github 上。要获取源代码，在终端中运行：（windows 下请自行安装 git）

```
git clone https://github.com/zzy7896321/holdem.git
```

评测系统需要Boost C++ Library，请自行下载编译。

holdem/client 目录下提供了 Makefile 和 Makefile-mingw。请 Makefile 中的 AI 变量改为 AI 实现的文件名，比如 AI 实现在 example.cpp 中，需要把 AI 设置为 example。

在 Linux 系统上，正常情况下

```
make
```

即可编译。

在 Windows (MinGW) 系统上，需要先设置好 MinGW GCC 的环境变量，将 Makefile-mingw 中 BOOST_PATH 改为 boost 的根目录。另外，由于 MinGW 的一些 bugs，可能需要对它提供的头文件进行修改，参见Link 1。

```
mingw32-make -f Makefile-mingw
```

在 Cygwin 上可能无法正常编译。

评测系统需要支持 C++11 的编译器，已知 g++ 4.8 或更高的版本可以使用。目前已在以下环境中成功编译：

- Linux Mint 15 32bit, g++ 4.8.2
- Linux Ubuntu 14.04 64bit, g++ 4.8.3
- Windows 7 32bit, MinGW g++ 4.8.1
- Windows 8 64bit, MinGW g++ 4.8.1
- (to be tested) Mac OS

运行 client，其中 <ip> 是 server 的 ip 地址，port 是端口

```
./client <ip> <port>
```

example.cpp 是一个简单的 UI，各个阶段会提示用户输入决策，仅仅作为测试程序。

2.2 实现 AI 的接口

Player 类中的以下函数需要实现，若额外的存储空间，可在 Player.h 中自行加入所需的成员变量。

- std::string Player::login_name();

  返回你的 AI 的名字，将在与服务器建立连接时调用（在调用 init 之前）。

- void Player::login_name(std::string name);

  服务器接受 name 作为你的 AI 的名字，这个名字可能与之前 login_name() 的不同。

- void Player::init();

  初始化函数。init 将在确认与服务器连接成功，收到玩家列表之后被调用。

- void Player::destroy();

  destroy 将在 Player 的析构函数中调用。

- decision_type Player::preflop();

  返回第一轮下注时 AI 的决定，可能在同一轮中被多次调用。

- decision_type Player::flop();

  返回第二轮下注时 AI 的决定，可能在同一轮中被多次调用。

- decision_type Player::turn();

  返回第三轮下注时 AI 的决定，可能在同一轮中被多次调用。

- decision_type Player::river();

  返回第四轮下注时 AI 的决定，可能在同一轮中被多次调用。

- hand_type Player::showdown();

  返回 AI 决定展示的 5 张牌，5 张牌应当在自己的 2 张手牌和 5 张公共牌中。

- void game_end();

  一局比赛结束的时候调用，AI 可以进行赛后统计，以便提供之后比赛使用的数据。

decision_type 定义为：

```
enum DECISION_VALUE { CHECK, FOLD, CALL, RAISE };
typedef std::pair<DECISION_VALUE, int> decision_type;
```

其中,CHECK，FOLD 和 CALL 作为决定时，第二个分量将被忽略。RAISE 作为决定时，第二个分量表示在跟注基础上，还要加注的大小。比如场上本轮最大下注为 4，自己现在已经下了 2, 那么 <RAISE, 4> 表示，跟注到 4 的基础上再加注 4，自己的下注将变为 8。可以调用 make_decision 函数返回适当的决定。比如，

```
return make_decision(CHECK);
return make_decision(CALL);
return make_decision(FOLD);
return make_decision(RAISE, 4);
```

hand_type 和 card_type 定义为：

```
typedef std::pair<char, char> card_type;
typedef std::array<card_type, 5> hand_type;
```

其中 card_type 两个分量分别表示牌的大小（2、3、4、5、6、7、8、9、T、J、Q、K、A）和牌的种类（S、H、D、C）。hand_type 是一个大小为 5 的数组, 存放 5 张要展示的手牌。

其他的定义详见 common.h。

比赛信息可以通过 query 成员的成员函数查询，目前提供以下查询函数：

函数原型	说明
整个游戏的信息，从 init 开始可以调用	
const std::string& name_of(int player);	返回编号为 player 的玩家的名字
int number_of_player();	玩家总数
int my_id();	自己的编号
int initial_chips();	初始的筹码数
const std::vector<int>& chips();	存放玩家筹码数的 vector，下标为玩家编号
int chips(int player);	返回编号为 player 的玩家的筹码数
单局游戏的信息，可以在 preflop 到 game_end 函数中调用	
int number_of_participants();	本局游戏中玩家数量
bool out_of_game();	自己是否已经出局
int dealer();	本局的庄家编号
int blind();	本局小盲注大小
const card_type* hole_cards();	返回存放自己的底牌的数组，大小为 2
const std::vector<card_type>& community_cards();	返回存有公共牌的数组，大小可能为 0,3,4，或 5
const std::vector<Pot>& pots();	返回存放有彩池的数组，一轮下注可能产生多个彩池，不同轮贡献者相同的彩池不合并，详见 pot.h
一轮下注的信息，仅在本轮调用有效	
const std::vector< std::pair<int, int> >& bets();	本轮下注的情况，pair 中两个分量分别为下注玩家编号和下注的大小，一个玩家可能多次出现

const std::vector<PLAYER_STATUS>& player_statue();	存放玩家下注状态的数组，PLAYER_STATUS 可能取值 NOT_ACTIONED, BET, CHECKED, FOLDED，下标为玩家编号
PLAYER_STATUS player_status(int player);	返回编号为 player 的玩家的下注状态
const std::vector<int>& current_bets();	存放本轮每个玩家的下注数量的数组，下标为玩家编号
int current_bets(int player);	返回编号为 player 的玩家在本轮下注的数量
单局比赛的统计信息，可以从上一局的 game_end 开始到下一局 game_end 之前调用	
const std::vector<HANDINFO>& hands();	返回存有上一局参与 showdown 的玩家手牌，HANDINFO 参见 common.h
const std::vector< <std::vector< std::pair<int, int> > >& won_chips_in_pots();	返回上一局每个彩池分配的情况，pair 的分量分别是玩家编号和玩家从该彩池赢得筹码的数量
const std::vector<int>& chips_won_in_last_game();	返回上一局游戏每个玩家赢得的筹码数

3 作业要求

- 实现一个能正常运行的 AI，并且有合理的运行时间
- 尽可能地赢得更多的筹码，并且能适应不同类型的玩家
- 在线测试时，会提供 server 的 ip 地址和端口，大约 8 至 9 人一组。
- 最终提交包括 AI 实现的源代码和一份报告，报告需要简要阐述你的算法和独到之处。评分会根据在线测试成绩、提交的代码和报告综合确定。