Computer Poker Competition

March 2013

The aim of this project is to submit an agent (.sh on linux) to compete in the Texas Hold'em. Please read through this instruction to prepare your agent.

Participants

Each team should consist 2-3 persons and submit the information including: Team Name, Team Leader, Team Members, Technique and references.

For example:

- **Team Name:** Alberta
- **Team Leader:** Alice
- **Team Members:** Bob, Mike
- **Technique:**
  The 2-player instant run-off program is built using the Public Chance Sampling (PCS) [1] variant of Counterfactual Regret Minimization [2]. We solve a large abstract game, identical to Texas Hold'em in the preflop and flop. On the turn and river, we bucket the hands and public cards together, using approximately 1.5 million categories on the turn and 900 thousand categories on the river.
- **References and related papers:**
  1. Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, and Michael Bowling. "Efficient Nash Equilibrium Approximation through Monte Carlo Counterfactual Regret Minimization" In AAMAS 2012
  2. Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. "Regret minimization in games with incomplete information" In NIPS 2008.

Rules

The Computer Poker Competition will take the form of **Heads-up Limit Texas Hold'em**. Details of the event rules are listed below.

Game	Heads-up Limit Texas Hold'em
Competition Format	Series of heads-up duplicate matches
Hands Per Match	3000
Stack Sizes	Infinite
Bet Sizes	10/20
Blind Sizes	5/10
Blind Structure	Reverse blinds, no ascending blinds
Showdown Mucking	No
Illegal Actions	Any illegal action is interpreted as a call

Getting started

The server code is provided in the “project_server” box. This code was developed and tested for use on Unix based systems. Though it may work on other platforms, there are no guarantees.

You will need standard Unix developer tools to build the software including gcc, and make.

===== Getting Started =====
* Building the code
The Makefile provides instructions for compiling the code. Running 'make' from the command line will compile the required programs.

* The programs
dealer -Communicates with agents connected over sockets to play a game example_player -A sample player implemented in C
play_match.pl -A perl script for running matches with the dealer

Usage information for each of the programs is available by running the executable without any arguments.

* Playing a match
The fastest way to start a match is through the play_match.pl script. An example follows:
$ ./play_match.pl matchName holdem.limit.2p.reverse_blinds.game 1000 0
Alice ./example_player.limit.2p.sh Bob ./example_player.limit.2p.sh

After play_match.pl finishes running, there will be two output files for the dealer and two output files for each player in the game:

matchName.err -The stderr from dealer including the messages sent to players
matchName.log -The log for the hands played during the match
matchName.playerN.std -stdout from player N
matchName.playerN.err -stderr from player N

Note, play_match.pl expects player executables that take exactly two arguments: the server IP followed by the port number. The executable must be specified such that it is either a path or the executable name if it can be found in your $PATH.

If you need to pass specific arguments to you agent, we suggest wrapping it in another script. play_match.pl will pass any extra arguments to dealer. Matches can also be started bycalling dealer and starting the players manually. More information on this is contained in the dealer section below.

* dealer
Running dealer will start a process that waits for other players to connect to it. After starting

dealer, it will output something similar to the following:

```
$ ./dealer matchName holdem.limit.2p.reverse_blinds.game 1000 0 Alice Bob
16177 48777
# name/game/hands/seed matchName holdem.limit.2p.reverse_blinds.game 1000 0
#--t_response 10000
#--t_hand 600000
#--t_per_hand 6000
```

On the first line of output there should be as many numbers as there are players in the game (in this case, "16177" and "48777"). These are the ports the dealer is listening on for players. Note that these portsare specific to the positions for players in the game.

Once all the players have connected to the game, the dealer will begin playing the game and outputting the messages sent to each player. After the end of the match, you should have a log file calledmatchName.log in the directory where dealer was started with the hands that were played.

Matches can also be started by starting the dealer and connecting the executables by hand. This can be useful if you want to start your own program in a way that is difficult to script (such as running it in a debugger).

==== Game Definitions ====

The dealer takes game definition files to determine which game of poker it plays. Please see the included game definitions for some examples. The code for handling game definitions is found in game.c and game.h.

Game definitions can have the following fields (case is ignored):

gamedef -the starting tag for a game definition
end gamedef -ending tag for a game definition
stack -the stack size for each player at the start of each hand (for no-limit)
blind -the size of the blinds for each player (relative to the dealer)
raisesize -the size of raises on each round (for limit games)
limit -specifies a limit game
nolimit -specifies a no-limit game
numplayers -number of players in the game
numrounds -number of betting rounds per hand of the game
firstplayer -the player that acts first (relative to the dealer) on each round
maxraises -the maximum number of raises on each round
numsuits -the number of different suits in the deck
numranks -the number of different ranks in the deck
numholecards -the number of private cards to deal to each player
numboardcards -the number of cards revealed on each round

Empty lines or lines with '#' as the very first character will be ignored

If you are creating your own game definitions, please note that game.h defines some constants for maximums in games (e.g., number of rounds).

Protocol Specification

This part describes the format of messages between the dealer program (server) and a player (client) Communication is over TCP, with clients connecting to the server (the client will be given a numeric address) using the ports printed out by the server. All messages are terminated by a carriage return and a new line ('\r\n' or ASCII characters 13 and 10, respectively.) Any message from the server which starts with '#' or ';' is a comment or GUI command, and may be safely ignored.

The first message from a player must be a version string
<version> := 'VERSION:2.0.0\r\n'

After this, the server will repeatedly send messages to the client giving their view of the match state, including states where the client is not acting, and the final state when the game is over. If the client is acting, they will respond by returning a message with the same state, followed by an action.
<serverMessage> := <matchState> '\r\n'
<matchState> := 'MATCHSTATE:' <position> <handNumber> <betting> <cards>
<position> := <unsigned integer> ':'
<handNumber>:= <unsigned integer> ':'
<betting> := { <limitBetting> } { <nolimitBetting> } ':'
<limitBetting> := (<round1LimitBetting>
{ | <round1LimitBetting> '/' <round2LimitBetting> ⋯ })
<roundXLimitBetting> = <limitAction>*
<limitAction> := <fold> | <call> | <limitRaise>
<fold> := 'f'
<call> := 'c'
<limitRaise> := 'r'
<nolimitBetting> := (<round1NolimitBetting>
{ | <round1NolimitBetting> '/' <round2NolimitBetting> ⋯ })
<roundXNolimitBetting> := <noLimitAction>*
<noLimitAction> := <fold> | <call> | <nolimitRaise>
<nolmitRaise> := 'r' <nolimitRaiseSize>
<nolimitRaiseSize> := <unsigned integer>
<cards> := <holeCards> <boardCards>
<holeCards> := <player1Cards> '|' <player2Cards> { '|' <player3Cards> ⋯ }
<boardCards> := <round1BoardCards> { '/' <round2BoardCards> ⋯ }
<playerXCards> := '' | <card> { <card> ⋯ }

<roundXBoardCards> := { <card> … }
<card> := <rank> <suit>
<rank> := '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'T' | 'J' | 'Q' | 'K' | 'A'
<suit> := 's' | 'h' | 'd' | 'c'
<clientResponse> := <matchState> ':' { <limitAction> } { <noLimitAction> } '\r\n'

Parts of the specification in brackets indicate game specific definitions. For example, in Texas Hold'em, there are four rounds with zero, three, one, and one board cards on the rounds respectively, so the actual definition would be
<round1BoardCards> := ''
<round2BoardCards> := <card> <card> <card>
<round3BoardCards> := <card>
<round4BoardCards> := <card>

The <position> field tells the client their position relative to the dealer button. A value of 0 indicates that for the current hand, the client is the first player after the button (the small blind in ring games, or the big blind in reverse-blind heads-up games.)

The <handNumber> is simply an identifier for the current hand. Clients should make no assumption about these values, other than that it will be unique across hands within a match, and that it will not change within a single hand.

The <betting> is a list of actions taken by all players. There is no distinction made between calling and checking, or betting and raising. In no-limit betting strings, the raise action includes a size, which indicates the total number of chips the player will have put into the pot after raising (ie the value they are raising to, not the value they are raising by.) Valid betting strings will be described in a later section.

<cards> is the complete set of cards visible to the player, given their <position>. <holecards> describes the view of the private cards, and the number of <playerCards> sections is determined by the game being player. For example, heads-up games will have two sections, 3 player ring games will have 3 sections, and so on. Each <playerCard> section will either be an empty string, indicating that the player does not know what those cards are, or a number of <card> strings determined by the game. A <card> is simply two characters, one for the rank and one for the suit. The <boardCards> description will also have a number of sections, one for each round, up to the current round as indicated by the <betting>. The number of <cards> in each section is fixed, and determined by the game. Note that if it ever becomes desirable to play a game with board cards in the first round, the protocol should probably be changed again to add a separator between the hole cards and board cards.

A valid <betting> string consists of a sequence of valid actions for successive acting players. Calling is always valid. Folding is only valid if the acting player would need to add money to the pot to call. To describe raises, it is worth making a distinction between raising by and raising to. Unless this document specifically mention otherwise, when it speaks of a raise size, it is talking about a raise to a value, using the term to mean the total number of chips the player will have put into the pot, including chips added in previous rounds. In contrast, raising by a value is adding that number of chips to the pot after calling the current bet.

A raise is valid if a) it raises by at least one chip, b) the player has sufficient money in their stack to raise to the value, and c) the raise would put the player all-in (they have spent all their chips) or the amount they are raising by is at least as large as the big blind and the raise-by amount for any other raise in the round. This description properly handles some odd cases where some players may be all-in. Informally, at the beginning of a round a player may bet as low as the big blind, and each subsequent raise must increase the bet by at least as much as the last raise. In limit betting games, there may also be a limit on the number of raises in a round, and any raise action over this limit is not valid.

An active player is one that has not folded, and is not all-in. Label the players from 0 to *N*-1, where *N* is the number of players in the game. After an action by player *p*, the next acting player is *p'* for the minimum *d*>0 such that *p'* is active and *p'* =(*p*+*d*) modulo *N*. That is, the next player around the table who has at least two valid actions. The first player in a round is specified by the game.

A betting round ends when all active players remaining have either called, or were the player which initiated the current bet. The game is over if the all betting rounds have finished, or if there are not at least two active players left in the game. If all players but one have folded, the remaining player wins the entire pot, and does not show other players their cards. Otherwise, the game has ended in a showdown and all non-folding players show everyone their cards.

Values in a showdown are determined by the rank of a players best hand, constructed from their hole cards and the public board cards. Folding players have a lower rank than any non-folding player. Every pot of money is split evenly between all participating players with the highest rank. There is a separate pot for every distinct amount *j* of money spent by a player (whether they fold or not), and a player is participating in the pot if they spent at least that much money (again, whether they fold or not.) The size of the pot is equal to the number of participating players multiplied by (*j* - the size of the next smallest amount spent by a player).

Examples

The rest of this part is examples of messages to and from a client in various games. Messages from the server are prefaced with S->. Responses from the client are prefaced with <-C.

Two player limit Texas Hold'em

```
S-> MATCHSTATE:0:0::TdAs|
S-> MATCHSTATE:0:0:r:TdAs|
<-C MATCHSTATE:0:0:r:TdAs|:r
S-> MATCHSTATE:0:0:rr:TdAs|
S-> MATCHSTATE:0:0:rrc/:TdAs|/2c8c3h
<-C MATCHSTATE:0:0:rrc/:TdAs|/2c8c3h:r
S-> MATCHSTATE:0:0:rrc/r:TdAs|/2c8c3h
```

```
S-> MATCHSTATE:0:0:rrc/rc/:TdAs|/2c8c3h/9c
<-C MATCHSTATE:0:0:rrc/rc/:TdAs|/2c8c3h/9c:c
S-> MATCHSTATE:0:0:rrc/rc/c:TdAs|/2c8c3h/9c
S-> MATCHSTATE:0:0:rrc/rc/cr:TdAs|/2c8c3h/9c
<-C MATCHSTATE:0:0:rrc/rc/cr:TdAs|/2c8c3h/9c:c
S-> MATCHSTATE:0:0:rrc/rc/crc/:TdAs|/2c8c3h/9c/Kh
<-C MATCHSTATE:0:0:rrc/rc/crc/:TdAs|/2c8c3h/9c/Kh:c
S-> MATCHSTATE:0:0:rrc/rc/crc/c:TdAs|/2c8c3h/9c/Kh
S-> MATCHSTATE:0:0:rrc/rc/crc/cr:TdAs|/2c8c3h/9c/Kh
<-C MATCHSTATE:0:0:rrc/rc/crc/cr:TdAs|/2c8c3h/9c/Kh:c
S-> MATCHSTATE:0:0:rrc/rc/crc/crc:TdAs|8hTc/2c8c3h/9c/Kh
S-> MATCHSTATE:1:1::|Qd7c
<-C MATCHSTATE:1:1::|Qd7c:r
S-> MATCHSTATE:1:1:r:|Qd7c
S-> MATCHSTATE:1:1:rr:|Qd7c
<-C MATCHSTATE:1:1:rr:|Qd7c:c
S-> MATCHSTATE:1:1:rrc/:|Qd7c/2h8h5c
S-> MATCHSTATE:1:1:rrc/r:|Qd7c/2h8h5c
<-C MATCHSTATE:1:1:rrc/r:|Qd7c/2h8h5c:c
S-> MATCHSTATE:1:1:rrc/rc/:|Qd7c/2h8h5c/Th
S-> MATCHSTATE:1:1:rrc/rc/r:|Qd7c/2h8h5c/Th
<-C MATCHSTATE:1:1:rrc/rc/r:|Qd7c/2h8h5c/Th:f
S-> MATCHSTATE:1:1:rrc/rc/rf:|Qd7c/2h8h5c/Th
S-> MATCHSTATE:0:2::9d7s|
S-> MATCHSTATE:0:2:r:9d7s|
<-C MATCHSTATE:0:2:r:9d7s|:c
S-> MATCHSTATE:0:2:rc/:9d7s|/5d2cJc
<-C MATCHSTATE:0:2:rc/:9d7s|/5d2cJc:c
S-> MATCHSTATE:0:2:rc/c:9d7s|/5d2cJc
S-> MATCHSTATE:0:2:rc/cc/:9d7s|/5d2cJc/3d
<-C MATCHSTATE:0:2:rc/cc/:9d7s|/5d2cJc/3d:c
S-> MATCHSTATE:0:2:rc/cc/c:9d7s|/5d2cJc/3d
S-> MATCHSTATE:0:2:rc/cc/cr:9d7s|/5d2cJc/3d
<-C MATCHSTATE:0:2:rc/cc/cr:9d7s|/5d2cJc/3d:f
S-> MATCHSTATE:0:2:rc/cc/crf:9d7s|/5d2cJc/3d
```

About the Project

If you have comments or suggestions, please post to the “Web Learning” in the discussion section.