POKER - example_player

This document gives a brief introcution to `example_player.cpp` and `example_player.py`, poker clients using simple random strategy.

They handle everything about socket or protocol for you. If you want to implement your client in other languages, these implementations can be a useful reference. If you decide to use C++ or Python3, fortunately, you can safely skip the communication parts.

example_player.cpp

Poker strategy should be implemented in `act` function. It has three arguments:

- `game`: a description of the game
- `state`: current game state
- `rng`: random state used to generate random numbers. You can use your own random generators.

For detail APIs of `game` and `state`, please refer to `game.c` and `game.h`.

game

The `game` object stores basic game configurations that may be useful for calculating poker strategy. The information will NOT change, and can be viewed as constants in our poker project.

state

The most useful part is `state`, which provides all the information about the current game state, including current private cards, board cards, money spent by both players, and action sequence.

```
typedef struct {
  State state;
  uint8_t viewingPlayer;
} MatchState;
```

```
typedef struct {
  uint32_t handId;

  /* largest bet so far, including all previous rounds */
  int32_t maxSpent;

  /* minimum number of chips a player must have spend in total to raise
     only used for noLimitBetting games */
  int32_t minNoLimitRaiseTo;

  /* spent[ p ] gives the total amount put into the pot by player p */
  int32_t spent[ MAX_PLAYERS ];

  /* action[ r ][ i ] gives the i'th action in round r */
  Action action[ MAX_ROUNDS ][ MAX_NUM_ACTIONS ];

  /* actingPlayer[ r ][ i ] gives the player who made action i in round r
     we can always figure this out from the actions taken, but it's
     easier to just remember this in multiplayer (because of folds) */
  uint8_t actingPlayer[ MAX_ROUNDS ][ MAX_NUM_ACTIONS ];

  /* numActions[ r ] gives the number of actions made in round r */
  uint8_t numActions[ MAX_ROUNDS ];

  /* current round: a value between 0 and game.numRounds-1
     a showdown is still in numRounds-1, not a separate round */
  uint8_t round;

  /* finished is non-zero if and only if the game is over */
  uint8_t finished;

  /* playerFolded[ p ] is non-zero if and only player p has folded */
  uint8_t playerFolded[ MAX_PLAYERS ];

  /* public cards (including cards which may not yet be visible to players) */
  uint8_t boardCards[ MAX_BOARD_CARDS ];

  /* private cards */
  uint8_t holeCards[ MAX_PLAYERS ][ MAX_HOLE_CARDS ];
} State;
```

There are also some functions that you may find helpful.

- `raiseIsValid`: check if a raise is possible
- `isValidAction`: check if an action is valid
- `numRaises`: number of raises in the current round
- `numAllIn`: number of players who are all-in
- `printState` & `printMatchState`: print a state to a string
- `printCard` & `printCards`: print cards to a string

example_player.py

Poker strategy should be implemented in `act` function.

GameState is a useful class that reads information from the match state string. After being initialized, it provides helpful members and functions for deciding a proper strategy. Please refer to the source code for details.