2013

U OF M PROGRAMMING CONTEST

Rules and Specification

The Game

The game is a card game that is played with a standard deck of 52 cards (no jokers). Only the rank is used, the suit does not matter. For simplicity, we will use the numbers 1 through 13 to represent the cards.

The game consists of a series of **hands**. Each hand may result in:

- A **point** being awarded to either one or zero players
- Ending the **game** due to an accepted challenge and a player winning the hand
- Ending the **game** due to one player accumulating enough **points**.

In a hand, each player will be dealt 5 cards from the deck. The winner of the previous hand will choose their action first. For the first hand, the first player will be chosen such that as we proceed through the games in a round, the initial lead will rotate for each player. The player with the lead may either choose to **play a card** on to the game field, or **issue a challenge**.

If a card is played, the opposing player chooses an action. If there have been no challenges in this game, the player may now issue a challenge themselves, or choose to play a card. Otherwise, they must play a card.

When both players have played a card on to the field, the player who plays the highest card **wins the trick**, and the lead for the next trick. If the two cards have the same value, neither player wins the trick, and the lead does not change.

In the event of a challenge being issue, the other player is forced to make one of two choices:

- **Reject** the challenge. If this is chosen, the hand ends immediately, and a point is awarded to the **challenger**. You do not get to see whatever remaining cards your opponent may have.
- **Accept** the challenge. If this is chosen, the hand is played out normally, and **the winner of this hand wins the game**.

As long as no challenge is rejected, play continues until all five tricks have been played. Any player may issue a challenge, so long as no challenge been issued in this hand.

Once all five tricks have been played, one of two things happens:

- If a challenge was issued, the player with the most tricks immediately wins this game. If Player 0 has one trick and Player 1 has zero, Player 0 wins; it is strictly the *most* tricks, not the majority of tricks played. If the players have a tied number of tricks, nobody receives any points, nobody wins the game, and play proceeds on as if the entire hand had not occurred.
- If a challenge was not issued, the player with the most tricks receives a **point**. If the players have tied the number of tricks they have taken, nobody receives a point.

If a player reaches 10 points, they win the game.

If no player has won, each player receives five new cards from the remaining unshuffled deck, and the next hand is played. Two standard-sized decks are used for this game. The deck will be shuffled when it no longer has sufficient cards to deal to the players. (That is, when 10 hands have been played, and 4 cards are remaining, all 104 cards will be reshuffled, and play will resume. We do not deal out the 4 cards, then shuffle the 100, then deal the remaining cards.)

Penalties

"To err is human. To screw up a million times per second, you need a computer."

Since we're playing with computer programs, we have to account for those extra-special computers-only failures. Your computer player may fail to make a legal move (play a card you don't have, make a currently-invalid move, etc.), fail to return something the server can understand, or take too long to respond. In the event that your computer program fails to make a correct move, you will be penalized in the following manner:

- If you have an opportunity to play a card, and you fail, your highest card will be removed from your hand, the lowest card will be removed from your opponent's hand, and your opponent will win the trick. If this was not the last trick, your opponent will also receive the lead for the next trick.
- If you are currently responding to a challenge request, an illegal move is equivalent to rejecting the challenge.

You begin each **game** with 2 seconds to make a move. For each of the first 100 moves you successfully make, you receive an additional 50 milliseconds. After that you will receive no more time. When a move is requested, you will also receive an *approximate* specification of how much time you have remaining. Bear in mind that due to network delays, general timing inaccuracies, etc., you can not count on being able to wait the full period of time that you receive to reply. Once you have run out of time, you will no longer be asked to make moves for that game. You will penalized each time.

The purpose of these times is to ensure that the final tournament is played at a reasonable pace, so we can not practically loosen them. While you are allowed to open port forwards to machines outside of the contest network so they can play, you will not receive extra time. This means it is probably practical to use machines that are also on the University network, but may make it impractical to use resources any more remote than that.

In order to prevent hammering on the server, you client must wait at least 5 seconds between connection attempts, or the server will deny your connection. For each attempt to connect before the 5 seconds is up, the timer is reset, so a client that is hammering on the server will never be accepted. All clients shipped by Barracuda will wait properly, so in order for this to happen, you must have broken the code. You will need to fix it to continue.

Rounds

In the general play period and during the tournament, games will be organized into **rounds**. Rounds consist of a series of games, which will be used to determine who won. The rules are as follows:

- If a player gets a 5 game lead over their opponent, they win the round.
- The first player to 21 games won wins the round.

Contest Structure

During the initial phase of the contest, the contest server will continuously match up players that are connected to the server and play them against each other using the rules specified above. The players will be ranked on an ongoing basis using a scoring system similar to the one used by the chess community.

In the final hour of open play, the game scheduling algorithm will stop taking into account whether or not you are connected, and all participating teams will be forced to play. If you are not online when you are scheduled to play, you will simply lose.

At the end of that hour, the scheduler will be turned off, and the standings at that point will be used to seed the final single elimination tournament. At this point you *must* at least be connected, or you will not have a place in the final standings.

Examples

This is an example of a normal **hand**. It ends with the score board showing the left player with 1 point.

1 1

Tricks: 0 Points: 0 Tricks: 0 Points: 0

Prev Next

Here is a hand in which the player issues a challenge, but the opponent rejects it. Note how the challenger is given the point:

12	6
9	6
8	4
8	2
1	1

Tricks: 0 Points: 0 Tricks: 0 Points: 0

Prev Next

Here is a hand in which a player issues a challenge, which the opponent accepts. Note at the end, the entire game is over.

12
9
8
8
1

6
6
4
2
1

Tricks: 0 Points: 0

Tricks: 0 Points: 0

Prev Next

Here is a hand in which a player issues a challenge, and the opponent errors out by losing its connection. Note how with the first error, the left player still wins the trick, even though the penalty removed a 1 from their hand and a 6 from the right player. It is *never* to your advantage to make an error.

12
9
8
2
1

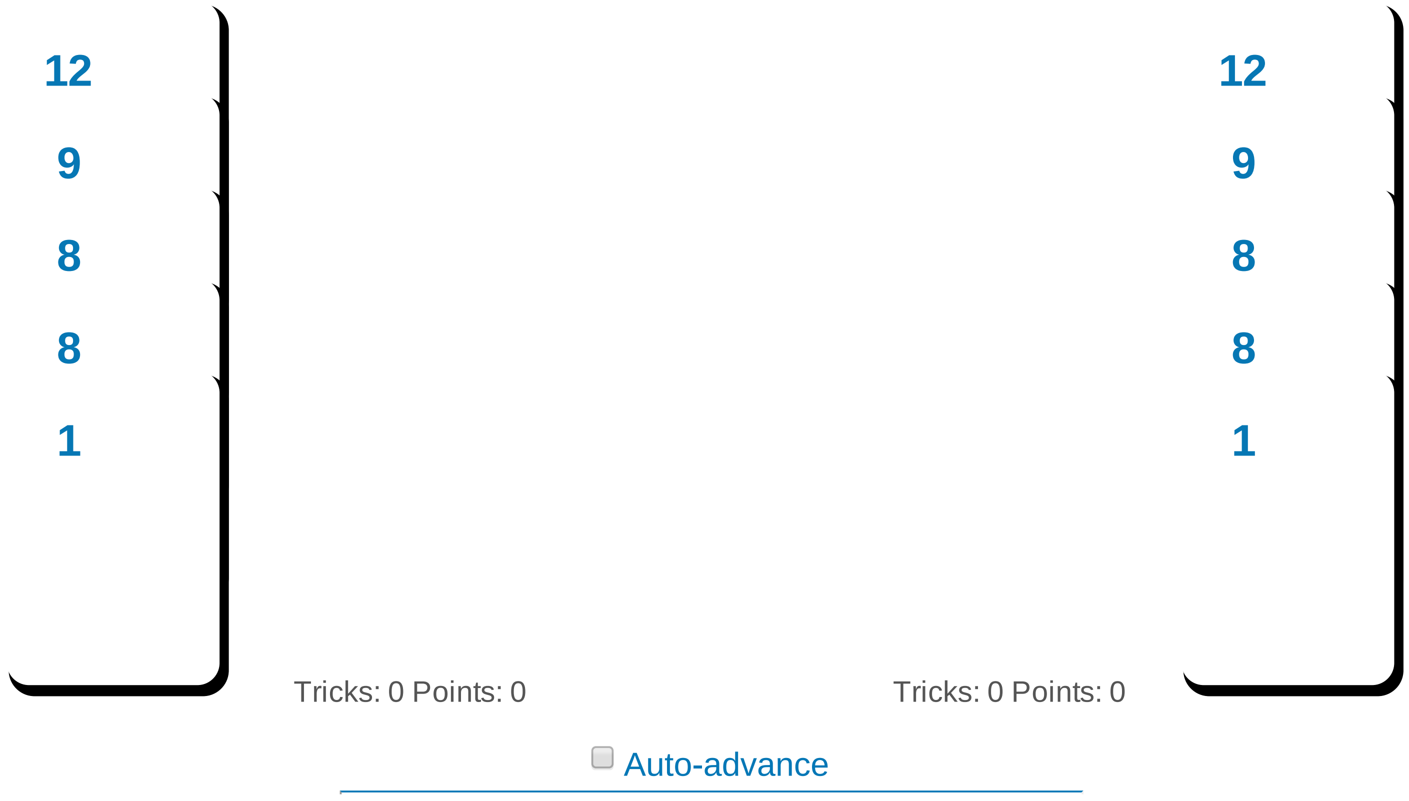
6
6
4
2
1

Tricks: 0 Points: 0

Tricks: 0 Points: 0

Prev Next

Here is a hand in which, by *amazing* coincidence, every trick is a tie, resulting in nobody getting any hands. If this were a real match, play would continue on as if the challenge had not occurred.



Prev Next

API

Your entry in the Barracuda Programming Contest must connect to the programming contest server with TCP, receive queries, and send back results, using a simple custom protocol.

This section will document the socket protocol, and what queries may be sent on it. If you use one of the Barracuda-provided skeletons, connecting the socket and speaking the raw protocol will be handled for you, and you need only worry about what the messages are saying. Due to the wide variance between how languages handle JSON and how their type systems work, **please read the README in your chosen language's archive file** for more information. Use of some language implementations may require you to know more than others about the details of this protocol.

Should you choose to use another language without a provided skeleton, Barracuda employees will try to provide as much help with the mechanical details as possible, but *caveat emptor*.

Binary Protocol

The socket protocol is a very simple one. The protocol consists of a series of messages. Each message is the following:

- Four bytes which are a network-ordered unsigned integer. This integer indicates how many of the following bytes to read to obtain this message.
- That number of bytes of a JSON object. (Technically JSON is always defined to be UTF-8 but we guarantee it will always consist only of the 7-bit ASCII subset.)

Your replies follow the same protocol.

Use The Exact Types

Note that the server is expecting *precisely* what is outlined here. If the protocol calls for a "number", it *must* be a JSON number. It can not be a string containing a number. Those using the Barracuda-provided code should have this already taken care of, those implementing new drivers should beware.

All messages the game server sends will be JSON objects that have a `"type"` field indicating their message type.

Connection

Open a standard TCP socket to port 9999 of this server. Your team affiliation will be identified by matching your apparent source IP to the source IP you gave on your team's page. If a match could not be located, the server will immediately send down a JSON message that looks like the following:

```
{"type": "error",
 "seen_host": "10.2.88.12",
 "message": "I don't know which team this connection..."}
```

The message will continue on beyond that. The `"seen_host"` field will contain the IP the server believes this connection is from. The TCP socket will then close.

If the server does recognize your IP, it will return an acknowledgment message. Its `"type"` will be `"greetings_program"`, the `"team_id"` field will contain the team ID the contest server believes you have, and other fields may exist which can be ignored.

If you lose connection during a game, if you reconnect quickly enough, you may receive a query or a message about a game in progress. You then have the opportunity to recover. However, please ensure you do not pound the contest server with endless connection attempts. The provided skeletons implement a wait period before attempting a reconnect.

Move Request

A move request is a JSON object with a `"type"` string of `"request"`. It contains the following elements:

- `"request"`: This will be a string containing one of two values:
  - `"request_card"`: It is your turn to play a card. Consult the `"can_challenge"` in the game state for whether or not you can challenge. Consult the `"card"` element of the game state to see what the card your opponent may have played is, if appropriate.
  - `"challenge_offered"`: A challenge has been offered to you.
- `"remaining"`: A floating point number indicating the approximate number of seconds you have to make this move.
- `"request_id"`: An integer that identifies this request. It must be sent back in your response as the `"request_id"` field.
- `"state"`: This contains a representation of the current state of the game. It has the following elements:
  - `"hand"`: This is an array containing integers representing your cards. Cards are numbered 1 through 13 inclusive. 13 is the highest card.
  - `"card"`: If a card is currently "on the table", this field will exist and have an integer representing the card. If there is no card on the table, this field will not be present.
  - `"hand_id"`: This begins at the integer 1 for the first hand, then is incremented for each hand played within a game. As that implies, these are *not* globally unique, just unique within a given game.
  - `"game_id"`: The identifier for this game. This can be used to look up the game on this website.
  - `"your_tricks"`: An integer representing the number of tricks you have taken so far in this hand.
  - `"their_tricks"`: An integer representing the number of tricks your opponent has taken so far in this hand.
  - `"can_challenge"`: A Boolean. If true, you may legally issue a challenge. If false, you may not.
  - `"in_challenge"`: A Boolean. If true, it indicates that this hand has had someone accept a challenge, and the winner takes the entire game.
  - `"total_tricks"`: An integer representing the total number of tricks played in this hand. (It may be less than `your_tricks + their_tricks` if there was a tie.)
  - `"your_points"`: An integer representing the number of points you have in this game.
  - `"opponent_id"`: This is the team ID of the team you are playing, as an integer. This will remain constant for a given team throughout the competition, regardless of team name or tagline changes.
  - `"their_points"`: An integer representing the number of points your opponent has in this game.
  - `"player_number"`: This will be 0 if you went first in the game, 1 if you went second. This is the same number that will be sent as the result of the `"by"` portion of the `"trick_won"`, `"hand_won"`, and `"game_won"` messages.

You must reply with a JSON object whose `"type"` is `"move"`, with the `"request_id"` copied from the request, and whose `"response"` is one of the following legal objects:

If the request was a `"request_card"`, you must reply with:

- An object with `"type"` set to `"play_card"`, and a `"card"` set to an integer representing the card in your hand you wish to play. (That is, if you want to play a 10, return a 10. This is *not* an index, it is the card number itself.)
- If you are allowed to offer a challenge, an object with `"type"` set to `"offer_challenge"`, and no other attributes.

If the request was a `"challenge_offered"`, you must reply with one of:

- An object with the `"type"` set to `"accept_challenge"` to accept it.
- An object with the `"type"` set to `"reject_challenge"` to reject it.

For example, the entire JSON message to reject the challenge would look like this (where `"request_id"` is `1`):

```
{"type":"move","request_id":1,"response":{"type":"reject_challenge"}}
```

Results

Once a player has made their move, information about the results of the move will be sent to the players. A given move will generate all the result information described below, as appropriate to the situation. Results will be a message with the `"type"` of `"result"`, and the actual result object in the `"result"` field. Additionally, at this level, all results will remind you of *your* player number with a field called `"your_player_num"` as an integer.

If the trick was won as the result of the last move, the result object will be of type `"trick_won"`, the `"by"` field will contain the player number of the winning player, and the `"card"` field will contain the *last card played in the trick*. This is *not* the winning card, this is the *last card played*. Both players will receive this.

If the trick was tied, the result object will be of type `"trick_tied"`, with no other fields. Both players will receive this.

If the hand was completed as the result of the last move, the result object will be of type `"hand_done"`. If the hand was won by a player, the `"by"` field will contain the player number of the winning player. If the hand was tied, there will be no such field. Both players will receive this.

If the game is now over as a result of the last move, the result object will be of type `"game_won"`, and the `"by"` field will contain the player number of the winning player. No further requests will be made with the game ID from this game.

If there was an error, the result object will be of type `"error"`, and the `"reason"` field will attempt to describe the error. Only the error-making player will receive this.

If none of these conditions was met, but the play was valid, the result object will be of type `"accepted"`, indicating a legal move. Only the player who made the move will receive this.

When receiving more than one of `"trick_won"`, `"hand_done"`, or `"game_won"`, they will arrive in that order. Note that it is possible to receive a `"hand_done"` with no `"trick_won"`, in the case of a rejected challenge.

Test Bot

If you would like to test your bot without it being "live", connect to the contest server, but use port 19999. After the connection message is sent out, you will be immediately placed into a game against a very stupid opponent.

Note this opponent will not test the full range of possible inputs your bot may receive; for instance, the test box will never issue a challenge.

Prepared Clients

Barracuda Networks uses a lot of open source software, and that means meeting that software in its own language. If you can name it, there's a decent chance we use it. Since this year we're handing out functioning clients in a functioning VM, this may be an interesting opportunity to try out a language you've had your eye on. Here's the set of prepared clients we've prepared this year, along with some editorial comments on whether you should consider using it. Lines of Code determined by `cloc` and a bit of judgment (does not include any shipped JSON libraries).

Language	Editorial Comments	LOC
C	The Grand Matron of modern programming languages. If you want this, you know it. But, if you're unsure... it's not a coincidence that it has the highest line count of any language here.	814
C++	C's bigger, badder son. If you want this, you know it. Again, though, it's not a coincidence that it's on the higher end of line count. If you're not sure, something with Garbage Collection for a contest like this might be a better idea.	494
ELisp	Wait, what? Emacs Lisp? One of our programmers got inspired. Works, but since emacs can't (quite!) do threading, locks your ~~editor~~ true OS while it runs.	107
Go	An up-and-coming language from Google, combining some of the best of the scripting world with pervasive use of interfaces with some of the best of the traditional statically-typed world. If you think you want to do something with multiple processors, one of the best choices on this list, but otherwise, you won't really be exercising Go's strengths much here.	157
Haskell	Forcibly expand your brain in directions you didn't know existed. If you've dabbled in Haskell already, this might be a good chance to try it on something a bit more real. I can't recommend this otherwise with a straight face though.	203

	The game-playing engine for the contest is implemented in Haskell.	
Java	The workhorse of the modern era.	430
JavaScript	Making a bid to escape the client, a battle-tested scripting language. The JS in JSON. This client runs on Node; as with Go, you may not necessarily have a lot of chance to run with Node's advantages, but it's a solid language for the contest.	124
Lua	The scripting language who can go toe-to-toe with C in performance via LuaJIT. Barracuda uses this quite extensively in some places where we need both performance and scripting.	40
Perl	The Grand Matron of scripting languages. An infrastructure language for Barracuda appliances. Rumors of its death are greatly exaggerated.	118
PHP	The Grand Matron of web scripting languages. (Maybe I need a new metaphor.) It also works just fine outside of the web server, it just doesn't make as much of a fuss as Node does. Also extensively used at Barracuda on our cloud products.	582
Python	Perl Done Right. Also on the curriculum at the University of Michigan, and almost the simplest client we have here, support code and all. Highly recommended. Even if you do not currently know it. The contest website is implemented in Python.	58
Scala	As Java becomes a bit *passé*, the JVM itself is getting new lease on life from the new generation of JVM languages, and Scala is one of the main contenders. Same recommendation as Haskell; if you've already dabbled, this is a great chance to dig in, but I can't recommend if you've never heard of it.	206
Tcl	The best scripting language you've never heard of. Arguably pioneered the idea of deeply embedding an event loop into the runtime of a language, putting it about 10-15 years ahead of its time. Also had a powerful built-in UI, though we don't use it here. A bit odd, though.	104

Virtual Machine Image (VirtualBox)

For those interested, we have prepared a VirtualBox VM image of Ubuntu Server 12.04 preloaded with the runtimes/libraries for the languages listed above. Following are instructions for deploying this VM:

Install VirtualBox.

Install Vagrant. On Windows 8, when installing Vagrant, if an error/smart screen appears, click "More Info" then

"Run Anyway".

Download the `Vagrantfile` for your platform:

- Linux/Mac
- Windows

Be sure to place the `Vagrantfile` in a new directory by itself (e.g., `~/Downloads/cuda-contest-vm/`). Make sure the file does not have an extension -- Windows may save it as `Vagrantfile.txt`, which is **not** what you want.

Change to the directory where you saved the `Vagrantfile`:

- Windows: `Start | Run | cmd` to get command prompt, then `cd` to the directory where you saved the `Vagrantfile`.
- Mac: `Applications | Utilities | Terminal` to get command prompt, then `cd` to the directory where you saved the `Vagrantfile`.

Run `vagrant up` in the same directory as the `Vagrantfile`. This will download the VM image and add it to VirtualBox. This may take a few minutes.

During installation, a directory named `contest-bots` that contains the sample bots will be created and will be shared between the host and the VM. You can update this code from either the host or the VM.

To get a shell on the VM:

- Linux/Mac: run `vagrant ssh` in the same directory as the `Vagrantfile`
- Windows: the VM will boot with a GUI
  - At the login prompt, log in as user `vagrant` with password `vagrant`.

This will log you into the VM as the `vagrant` user. The sample bot code is contained in `/vagrant/contest-bots`.

To stop the VM, run `vagrant halt`.

General

- Your program will not be asked to participate in more than one game at a time, so a single-threaded implementation will be fine. (This year, even requesting a run against a test bot will be in a separate network connection, probably in a different process.)
- In general, in fairness for all contestants, there will be little "mercy" given for non-functional bots. If your turn comes up in a game **or the final tournament** and your bot is unreachable by our driver for any reason other than the plain malfunctioning of our driver or the network as a whole, your turn may be scored as timed out. While modifications to your bot can be made at any time, we **strongly strongly recommend** against doing so during the final round of the contest. However, you are free to modify your bot at any time at your own risk, even mid-game. Emphasize *at your own risk*. People have indeed modified their functional bots into non-functional bots during the tournament at previous contests.
- Yes, it is perfectly legal to hold your AI in reserve and register it only for the single-elimination, prize-winning

tournament portion of the competition. However, be aware that you will be facing down the people who have been testing their bots against each other and tuning and tweaking them in real "battle" conditions. Your choice.