Internet Program Competition

Paul E. Utgoff

Technical Report 95-67 (revised)
September 15, 1995

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

Telephone: (413) 545-4843
Net: utgoff@cs.umass.edu

Contents

Abstract

This report shows how programs can engage in competition over the internet. Such an approach makes it possible to compare program performance without a human interface, without divulging source code, and without the need for a central server. Source code (publicly available) is described for Othello, Checkers, and Hearts. The source for a complete legal player program is included with each.

1 Introduction

In the computer game-playing community, a fundamental and persistent question is 'which program is best?' This curiosity is due largely to genuine lack of knowledge about which combinations of which techniques are better than others. For those with a competitive spirit, bragging rights also play a role. One aspect of game-playing is that there is always a definitive test for superiority.

Surprisingly, it has not been easy to test one program against another. Authors generally wish to keep their code private, at least until it becomes worth divulging. This need for privacy can lead to awkward arrangements that may even include people (usually the program authors) in a makeshift interface for exchanging the moves of the programs. In addition to the inconvenience, this kind of setup requires that the program authors be present and participate during the competition. Such impediments tend to make contests a rare event. This is unfortunate because it hinders each scientist's ability to measure the quality of the ideas that have been implemented in the program.

A second arrangement that sometimes occurs is for one author to make his source code available to another person, who merges the various pieces of code into a single program that pits the opponents against each other automatically. This eliminates the need for human involvement, but it produces at high cost a highly specialized program with a short life-expectancy.

A third arrangement that has appeared recently is the creation of a central game server. Programs are deposited at a central location on the internet, and are runnable through a Web interface. This too eliminates the need for humans to be involved in program communication, but it creates a computational bottleneck and it does not allow a program author to keep the code private. More recently, some have experimented with getting programs to communicate with one another over the Web through a central server.

This report presents a distributed internet model that does not require human participation, does not require divulging source or object code, and does not require a central server. The code that implements it is publicly available via anonymous ftp (see Section 3 below).

2 Internet Model

Each competitor runs as a separate *player program* on a node (computer) of the internet. In addition, a single *administrator program* also runs separately on an internet node. Some or all of these programs can be runing on the same internet node. Each player program communicates with the administrator program, whose job it is to tell each player program what it needs to know, and to query each player program for the information that it needs to administer the game. This model handles games both of perfect and imperfect information. Player programs do not communicate with each other directly. Instead, all information is exchanged through the administrator program.

The administrator sets up the communication channel for each player program. A player program needs only to receive messages from the administrator by reading from the provided file descriptor or to send messages to the administrator by writing to the provided file descriptor. The messages that are sent and received follow a simple protocol so that sender and receiver each

interpret them identically. The messages are short, providing efficient communication.

Any person (or program) can invoke a competition of the specified player programs, wherever those player programs reside on the internet. No one else is needed. The player programs are always available. No code is divulged. No central server is required. In addition, interactive play is possible (depending on the implementation) by specifying that a particular player's moves will be obtained interactively from the person who is running the administrator. This model requires that all other players be programs.

Each internet node on which a player program is to be accessible must run a *listener program* as a daemon. These mechanics are entirely hidden from the author of a player program, but are mentioned here for the curious. The administrator that is invoked on one node makes contact with the listener on the internet node for a player program, and asks that listener to start the indicated player program. If successful, the administrator program is supplied with a file descriptor that is connected to the specified player program. Otherwise, the failure is detected and appropriate messages are displayed to the invoker of the administrator.

3 Software for Three Internet Games

This section describes how the internet model has been instantiated for each of Othello, Checkers, and Hearts. Othello and Checkers are two-person perfect-information games, so there are many similarities. Hearts is a three-to-five-person imperfect-information card game.

The software and this report are available via anonymous ftp to `ftp.cs.umass.edu` on directory `/pub/internet-games`. Each game distribution includes source code for the administrator, the listener program, and a working sample player program. A program author can get started most quickly by copying the sample player program, and replacing its move selection strategy (legal but otherwise random) with a better one. When done this way, a program author can ignore a great deal of what is described here, including the details of the message protocol.

For useful details about writing a player program, consult the sample player program that is included in the code distribution. Here are some helpful items in case you want to start from scratch or use some additional information.

The file descriptor that the player program is to use is passed to the player program as its first command-line argument. To find out what it is, the player program should do the equivalent of `fd = atoi(argv[1]);` to define it as an integer.

One can do low-level read and write calls on the file descriptor. Some programmers prefer the calls of the standard i/o library. To set up the file pointers, the program should do the equivalent of `wfp = fdopen(fd,"w"); rfp = fdopen(fd,"r");`. A player program that uses this kind of stream i/o must be sure to `fflush(wfp)` whenever the stream is to be sent to the administrator.

Because a player program will be started by the listener program running on the same node, one should be wary of writing a log file, because the player program is started by the listener program and therefore runs as the user who started the listener. If you want a log file, one way to do this is to write it on /tmp, and then have your program mail the contents to you (see sendmail(8)). Alternatively, you can set the su bit (see chmod(2)) so that your player program always executes with your read/write protections, allowing the program to write a log file on your directory no matter who executes it.

Each of the next three subsections is written independently. A reader with general interest can pick any one of them. A reader with specific interest in one game can safely ignore the other sections. Reading more than one section may become tedious.

3.1 Othello

This section provides the specific information for the internet Othello code. Much of it can be ignored by starting with the sample player program included in the code distribution. It is included here for the sake of completeness.

The administrator and player programs communicate by exchanging messages as defined here. A message is a sequence of characters. The possible messages from the administrator to a player are:

`b`	you play as black.
`w`	you play as white.
`m####`	your move, you have `####` total seconds left. The total seconds part of the message is the printing characters of a 4-digit positive integer using leading '0' characters as necessary. leading
`orc`	your opponent moved at row `r` column `c`. The `r` is a character in the range 'a' to 'h'. The `c` is a character in the range '1' to '8'.
`z`	your opponent has no move.
`?b`	black player is confused, bye.
`?w`	white player is confused, bye.

The possible messages from a player to the administrator are:

`+`	contact established, I am ready to play.
`mrc`	I move at row `r` and column `c`. The `r` and `c` have the same interpretation as above.
`z`	I have no move.
`?`	I am confused, bye.

One can invoke the administrator, called `o-admin`, with a variety of command-line options. Most important are the specifications for the two player programs. The first is for the black player program and the second is for the white player program. Each is of the form `user.subdir@netaddress`. Any internet node that is to be accessible by the administrator must be running the listener program `o-listen` as a daemon. Any user can start it. Ideally it would be started automatically at boot time.

The administrator program contacts the listener program on the indicated internet node, and asks it to start the indicated player program. The listener program looks up the specified user's home directory, and then starts the program named `/home/othello/subdir/player`, if possible. Here, `home` is the indicated user's home directory, and `subdir` is that part of the player program specification. This naming scheme prevents an outsider from executing an arbitrary program on an internet node that is running the listener.

It can become tedious to type player program specifications. One can instead define an alias in one's `.othellorc` file, and use the alias instead. The `.othellorc` included in the distribution illustrates the simple format `alias aliasname playerspec` for each line. The special name `tty` cannot be aliased. When `tty` is used to specify a player, the administrator will obtain that player's moves from the user who invoked `o-admin`.

The are five command-line options:

`-t#` Set the total minutes for each side to # minutes. The default is five (5) minutes total per side. The minimum is 1, and the maximum is 60. A player program must manage its own time. A player program that exceeds its allotment loses, though the administrator currently takes no special action.

`-a` Produce a postscript file depicting every board position during the match (one file per position).

`-p` Produce a postscript file depicting the move history and final position for the match (one file).

`-l` Produce a text summary of the game.

`-q` Write nothing to the screen during the game. The `-q` is ignored if there is a `tty` player.

The initial total number of seconds available to each player for all moves during the game is available from argv[2]. One would need to do the equivalent of `game_seconds = atoi(argv[2]);`. The administrator tells the player how much time it has remaining each time it requests a move. Some programs need to know the total time allotment before play begins.

The opponent's login name is available in `argv[3]`, the opponent's program name is available in `argv[4]`, and the opponent's host name is available in `argv[5]`. These have a variety of uses, but a player program may ignore these safely.

3.2 Checkers

This section provides the specific information for the internet Checkers code. Much of it can be ignored by starting with the sample player program included in the code distribution. It is included here for the sake of completeness.

The administrator and player programs communicate by exchanging messages as defined here. All communication takes place through the administrator. A message is a sequence of characters. In the message desciptions below, note that the parentheses and commas are part of the message. The possible messages from the administrator to a player are:

`pb` you play as black (x).

`pr` you play as red (o).

`q(#)` tell me your move, you have # total seconds left, where the # consists of the printing characters of a positive integer. The # is of variable width.

`o(s1,...,sn)` your opponent moved its piece as indicated from `s1` to `sn`. The `s#` are two characters each. The first character is a letter in the range 'a' to 'h' specifying the row, and the second char is a digit in the range '1' to '8' specifying the column.

`cb` black player (x) is confused, bye.

`cr` red player (o) is confused, bye.

`eb` black wins, bye.

`er` red wins, bye.

`es` opponent not started, bye.

`ed` game drawn, bye.

`ep` network connection broken, bye.

The possible messages from a player to the administrator are:

`+` Contact established, I am ready to play. (Player program must send this once, when it is first started.)

`m(s1,...,sn)` I move my piece as indicated from s1 to sn. See o() above for specification of each `s#`.

`?` I am confused, bye

One can invoke the administrator, called `c-admin`, with a variety of command-line options. Most important are the specifications for the two player programs. The first is for the black player program (x) and the second is for the red player program (o). Each player specification is of the form `user.subdir@netaddress`. Any internet node that is to be accessible by the administrator must be running the `c-listen` as a daemon. Any user can start it. Ideally it would be started automatically at boot time.

The administrator program contacts the listener program on the indicated internet node, and asks it to start the indicated player program. The listener program looks up the specified user's home directory, and then starts the program named `/home/checkers/subdir/player`, if possible. Here, `home` is the indicated user's home directory, and `subdir` is that part of the player program specification. This naming scheme prevents an outsider from executing an arbitrary program on an internet node that is running the listener.

It can become tedious to type player program specifications. One can instead define an alias in one's `.checkersrc` file, and use the alias instead. The `.checkersrc` included in the distribution illustrates the simple format `alias aliasname playerspec` for each line. The special name `tty` cannot be aliased. When `tty` is used to specify a player, the administrator will obtain that player's moves from the user who invoked `c-admin`.

The are three command-line options:

`-t#` Set the total minutes for each side to # minutes. The default is five (5) minutes total per side. The minimum is 1, and the maximum is 60. A player program must manage its own time. A player program that exceeds its allotment loses.

`-l` Produce a text summary of the game.

`-q` Write nothing to the screen during the game. The `-q` is ignored if either of the players is `tty`.

The initial total number of seconds available to each player for all moves during the game is available from argv[2]. One would need to do the equivalent of `game_seconds = atoi(argv[2]);`. The administrator tells the player how much time it has remaining each time it requests a move. Some programs need to know the total time allotment before play begins.

The opponent's login name is available in `argv[3]`, the opponent's program name is available in `argv[4]`, and the opponent's host name is available in `argv[5]`. These have a variety of uses, but a player program may ignore these safely.

3.3 Hearts

This section provides the specific information for the internet Hearts code. Much of it can be ignored by starting with the sample player program included in the code distribution. It is included

here for the sake of completeness.

The administrator and player programs communicate by exchanging messages as defined here. A message is a sequence of characters. The possible messages from the administrator to a player are:

`injs`	There are `n` players. You are player `j` (players are indexed 0 through `n-1`). The game will be over when at least one player is above `s` or below `-s` points. The arguments `n`, `j`, and `s` are each a single positive 8-bit quantity indicating a value between 0 and 255.
`cc`	You are dealt card `c`. Cards are indexed 0 through 51, and `c` is in this range of 8-bit values. The values 0 through 12 correspond to the 2 through Ace of spades. Similarly, values 13 through 25 correspond to the 2 through Ace of hearts, values 26 through 38 correspond to the 2 through Ace of diamonds, and values 39 through 51 correspond to the 2 through Ace of clubs.
`gnp`	Pass `n` cards to player `p`. The player program must pick `n` cards to pass to player `p`, and tell the administrator what those cards are (see below).
`pcp`	You are passed card `c` from player `p`.
`yt`	It is your play (lead or follow), you have `t` seconds left for this hand. Unlike most of the other arguments, `t` is three printing characters of a positive value, with leading '0' characters as needed.
`tpc`	Player `p` has just played card `c`.
`kc`	You collect card `c` from the kitty.
`upd`	The total score of player `p` goes up by `d` points.
`dpd`	The total score of player `p` goes down by `d` points.
`?p`	Player `p` is confused, everyone is exitting, bye.

The possible messages from a player to the administrator are:

`+`	Contact established, I am ready to play.
`pc`	I pass card `c`.
`dc`	I play card `c`.
`?`	I am confused and am exitting, bye.

One can invoke the administrator, called `h-admin`, with a variety of command-line options. Most important are the specifications for the three-to-five player programs. Each is of the form `user.subdir@netaddress`. Any internet node that is to be accessible by the administrator must be running the `h-listen` as a daemon. Any user can start it. Ideally it would be started automatically at boot time.

The administrator program contacts the listener program on the indicated internet node, and asks it to start the indicated player program. The listener program looks up the specified user's home directory, and then starts the program named `/home/hearts/subdir/player`, if possible. Here, `home` is the indicated user's home directory, and `subdir` is that part of the player program specification. This naming scheme prevents an outsider from executing an arbitrary program on an internet node that is running the listener.

It can become tedious to type player program specifications. One can instead define an alias in one's `.heartsrc` file, and use the alias instead. The `.heartsrc` included in the distribution illustrates the simple format `alias aliasname playerspec` for each line. There is not currently a mechanism for playing any of the hands interactively.

The are five command-line options:

`-t#`	Set the total minutes per hand for each side to # minutes. The default is five (5) minutes total per player. The minimum is 1, and the maximum is 10. A player program must manage its own time. A player program that exceeds its allotment loses the match, though the administrator currently takes no action.
`-s#`	Set the match limit to # points. The default is 100. The minimum is 10, and the maximum is 1000. The match is over when a player reaches the match limit in either the positive or negative direction.
`-l`	Produce a text summary of the game.
`-q`	Write nothing to the screen during the game.
`-h`	Show a summary of the previous hand played before proceeding with the next.

There is additional information available in the arguments supplied to a player program. A player program can safely ignore this information. The total seconds allowed per hand per player is provided in `argv[2]`. The total number of players is given in `argv[3]`. Finally, starting at `argv[4]`, every three arguments are the login-name, program-name, and host-name for all the players. So, `argv[4]` is the login-name of player 0, `argv[5]` is the program-name of player 0, `argv[6]` is the host-name of player 0, `argv[7]` is the login-name of player 1, and so on.

4 Authentication

The administrator passes opponent-identifying information to each player program. Because the administrator source code is publicly available, it is possible that someone could create a version that misidentifies the players. This is quite unlikely. However, results that look suspect are easily testable since anyone can run any player program from his own local correct administrator program.

5 Joining the Community

Tell your fellow competitors where they can find your program so that they can add it to their file of aliases (.othellorc, .checkersrc or .heartsrc). Send a message to utgoff@cs.umass.edu to be added to the aliases file that is part of the distribution. Another way to become known is to play programs listed in the distributed aliases file. Because a player program is told who it playing, the opponent can become aware of the newcomer (especially if the newcomer wins).

6 Machine Load

There is no control over the number of processes that can be running at one time on one machine. It is up to each author to control the load on his/her machine as desired. Experience to date has shown this not to be a problem, but that may change. It is natural to guess that the best programs will be challenged often. A player program can be written so that it exits immediately if it has a reason not to play at that moment.

7 Standard Code

For the three internet games described here, it could become chaotic for different people to create nonstandard versions of the internet software. The best course of action, at least for the near term, is to send your suggestion (or code) to `utgoff@cs.umass.edu`. Appropriate acknowledgment will be given. If someone follows this model for a different game, I shall be glad to provide a pointer to that software. I intend to start a Web page soon.

8 Historical Notes

The basic idea for the distributed model of administrator and player programs originated in 1986 when I offered a seminar on computer game-playing. The Othello and Hearts code were written at that time to use socket-based local IPC. These programs were modified in 1993 to use socket-based internet IPC. When the seminar was offered in 1988, Backgammon was added, using the local IPC model. This code has fallen into disuse, and has not been modified to use internet IPC. The Checkers code was written during the summer of 1995.

Acknowledgements

Jeff Clouse provided helpful comments on an earlier draft of this document.