The Caia Project
Explained for the games Lamistra and Rolit

The Caia Project is the result of the need to have software to play competitions between programmed client players of strategic games[1] from their own homes. That software is now available under a UNIX[2] environment.

Introduction

The Dutch Olympiad in Informatics (NIO) has organized competitions between client player programs since 1996. In the beginning a match was played by starting a client player (player) program each time a move was needed. The history of the match was put in a text file that was updated by the jury software with the opponent's last move to get the next one. When the organization of the NIO introduced the CodeCup[3] in 2002, it also introduced a new protocol to play games. In the new situation both players have to be started only once. By using the stdin and stdout[4], the jury software reads a move from one player using its stdout and prints it to the other player, using its stdin, after having done the necessary checks. The new method of playing games has made it more difficult for the programmers to organize competitions at home. Under Windows[5] it is practically impossible to do this faultlessly; under UNIX it is fairly difficult but to be done.

Description of the project

Jaap Taal and Marcel Vlastuin developed a home edition of the jury software and published it as open source. The project Caia[6] consists of several components:

- The program *caiaio*, which is written in the language C. This is the main program that takes care of the input and output and is ready-to-use. Caiaio stands for: Caia-input-output.
- Programs that have to be written by the user are: a manager, a referee and, of course, the players themselves. The referee and the players are game dependent; the manager is not. The programs can be written in any language, the project has been tested for C, Pascal and Java players.
- The protocol that explains how the referee, the manager and the players can communicate with the caiaio and the others.
- An example of how to play a game between four Rolit players.
- An example of how to play a game between two Lamistra players.

The content of this project can be downloaded from the CodeCup website.

---

[1] Strategic games like Chess, Checkers, Othello and Rolit. Rolit is played by four players.

[2] Including Cygwin (see www.cygwin.com).

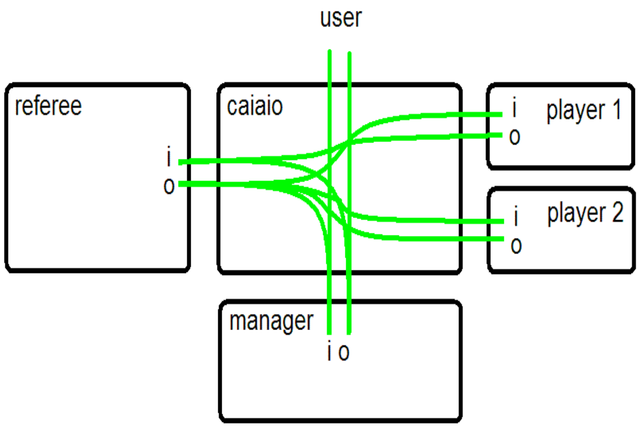
[3] See www.codecup.nl.

[4] The names stdin and stdout refer to virtual files from which a program can read and write. Under normal conditions it can read from the stdin using the keyboard of the computer and write to the stdout by printing text to the screen of the monitor.

[5] Reasonable successes can be achieved if Cygwin is used.

[6] Caia is actually the third project. It started with Atlas and Bubbles. With Bubbles, competitions were played with the CodeCup game Luckywords. The letters aia in Caia are the vowels from the CodeCup game Lamistra.

Overview of Caia



The manager[7]

To get things working the user must start the program *caiaio*. The first thing *caiaio* will do is execute the manager. In general, the manager takes care of games that must be played between different players in order to find out which is the best one: i.e. it runs a competition. The manager must be written and compiled by the user.

The caiaio is made in such a way that the players can identify themselves. If a player is started by the manager, *caiaio* waits for the first output that is sent by the player using its stderr. That output, called firsterror, is sent to the manager. In this way a player can inform the manager who and what sort of client player it is. If the manager is playing a competition between different players it can make use of a database. Games that have been played before, according to the database, can be omitted. However, if a player is producing random moves in some way, the game must be replayed. It is possible to play more games if one of the players is a random player so that the average result can be stored. The firsterror can be denied and is not compulsory. The players in use are fully compatible for both the CodeCup competition as well as for Caia competitions.

If the manager decides that a game must be played, the referee is launched in order to have it played. As explained, the players have already been started by the manager. After the game, the players are stopped independently and the referee's report must be sent to the manager. The referee must stop itself.

The user can provide the manager with information about how to do things by using a text file, the name of which can be included as an argument from the command line.

The referee

If the referee is started, it communicates primarily with the players and sometimes also with the caiaio. Like the manager, the referee must be written and compiled by the user. It is up to the user to make a simple version of a referee or not. Nevertheless it is wise to

[7] In the last distributions a far more advanced manager is available. You can use *competition* to run competitions very easily. How this can be used is described in the document that comes with the distro of the game.

include a checker for the moves generated by the players. The checker should be made independent of the players for debugging purposes.
What they should be sent depends on the protocol being used for the players. Most likely, for the white player, the following will be sent: “1 Start”. Its answer is sent back and then, most likely, the referee will send ”2 d4-d5” for the black player. The caiaio omits the first part of the messages and will send “Start” to player 1 and ”d4-d5” to player 2.

If something goes wrong with a player, the referee will be informed by the caiaio. Instead of an answer from the player concerned one of the following four messages will be sent to the referee:

- <caiaio:_player_stopped>
- <caiaio:_player_crashed_%s> (%s is replaced by the type of crash)
- <caiaio:_player_timeout>
- <caiaio:_player_sends_”\n”>.

When a game is over, the referee can ask the caiaio to send the ”playing time used” of both players. With that the referee can make up a complete report for the manager.

The players
The players must be written using the protocol for them. It is possible to make use of the firsterror to identify a player. In that case the player should send – unasked – an ID to its stderr. Don’t forget to flush it, because the caiaio will only wait a limited amount of time for the firsterror. The firsterror will be received 10 to 200 milliseconds after executing the player, depending on the size of the player, the hardware and operating system of the computer concerned. It is safest to use 1000 milliseconds to be sure the firsterror is seen.

The players can print information to their stderr. The caiaio scans that information and prints it to the screen (if it is in the debug mode) and to a log file (if wanted).

How to get and install the Caia project?
The Caia project can be used and achieved by getting the correct tarball[8] from the CodeCup website (www.codecup.nl). It installs perfectly on Linux as well on OSX and Cygwin platforms.

First you have to download the tarball *caia_lamistra_rolit.tar.gz* and put it in any directory. Then you must extract the source code of Caia and the binaries of Lamistra and Rolit in that directory. Execute from the command line:

```
tar -xzf caia_lamistra_rolit.tar.gz
```

How to compile the caiaio?
The source files in *caia_install_lamistra_rolit/caiaio/* need to be compiled and linked. Then the executable and some other files must be copied. You can do that by executing from the command line:

```
caia_install_lamistra_rolit/install_game.sh <namegame>
```

There are two games to install. For <namegame> you can choose out of two possibilities:
- lamistra (the CodeCup 2005 game is included as an example)
- rolit (an example game for four players)

The scheme on the next page shows the directory structure of the two distributions after being installed.

[8] You will get extra information of how to build a manager and a referee; examples of extra games are included.

The folder *caia/* is extracted in the home directory of the user.

caia/	
caia/lamistra/	*caia/rolit/*
caia/lamistra/bin/	*caia/rolit/bin/*
manager.txt	manager.txt
caiaio*	caiaio*
manager*	manager*
referee*	referee*
javawrapper*	javawrapper*
jarwrapper*	jarwrapper*
d11*	rolit_niv_2*
versie6VBLCS*	rolit_niv_3*
	rolit_niv_4*
caia/lamistra/src/manager/	rolit_niv_5*
manager.cc	
Makefile	*caia/rolit/src/manager/*
	manager.cc
caia/lamistra/src/referee/	Makefile
referee.cc	
Makefile	*caia/rolit/src/referee/*
	<cc files>
caia/lamistra/src/players/	Makefile
d11.pas	
versie6VBLCS.cc	*caia/rolit/src/players/*
Makefile	Makefile
	<cc files in directories>
caia/lamistra/refereelogs/	
	caiai/rolit/refereelogs/
caia/lamistra/playerlogs/	
	caia/rolit/playerlogs/

The items in cursive are folders; the items marked with an asterisk are executables. The log files of the players and the referee are put in *playerlogs/* and *refereelogs/*.

The programs of the manager or other executables in the directories can be built using the command *make* from the makefile; an update of the executable is automatically put into the *bin/* directory of that game.
The makefile from the players directory builds all C and Pascal client players in that directory and updates them into the *bin/* directory of that game as well. The C and Pascal compilers need to be installed by the user himself.

The source files and the makefile of the caiaio are not copied.
You can find them in the folder *caia_install_lamistra_rolit/caiaio/*.

How to start the caiaio for a game?
The program *caiaio* must be started from the command line in the *bin/* directory of a game:

```
./caiaio [-d] [-f <information>] [-m <executable>]
```

The three flags are optional:

`-d` Setting this flag causes the caiaio to print debug information to its stdout: the debug information will be printed to the screen. You might find this very useful because you can see what the caiaio actually is doing.

`-f` With this flag set, the string <information> will be passed as an argument when starting the manager by the caiaio. This option is included in the protocol in order to make it possible for you to provide information for the manager. The string can refer to an information file; that's up to the user.
Remind that you cannot use the stdin and stdout[9] to communicate with your manager: the caiaio is doing that! It is possible though, to print information from the manager to the screen using its stderr[10].

`-m` With this flag set, the executable with the name <executable> will be started by the caiaio as the manager. In this way it is possible to use different managers. By default the program *manager* will be started by the caiaio.

The protocol for the manager and the referee
It is preferred that when the referee communicates with a client player the referee is locked to the caiaio. The lock secures a good time measurement of the playing time of the client player. In some occasions the manager or the referee must not be locked. A locked program must always be unlocked after some time in order to enable the caiaio to read the stderr of the client players. The stderr of the manager and the referee will not be read by the caiaio.

Command:	Expected immediate response:
`I lock`	`lock_ok` This response, sent by the caiaio, must be read before continuing.
`I unlock`	No response!

[9] In C this is done with the functions scanf and printf; in Pascal with the functions ReadLn and WriteLn. In order having the games executed fast by the caiaio, the printed information has to be flushed. In C this is done with fflush(stdout), in Pascal with Flush(Output) and in Java with System.out.flush().

[10] In C you can do this with fprintf(stderr, <format string>, <variables>) followed by fflush(stderr); in Pascal with WriteLn(StdErr, <information>) followed by Flush(StdErr); in Java you can flush with System.err.flush().

The protocol from the point of view of the manager

Command:	Lock?	Expected immediate response:
`I cpu_speed <gamepc> [<mypc>]` Informs the caiaio about the cpu speed of the competition computer and the cpu speed of your computer (it will be estimated if omitted). If the command is omitted the caiaio assumes the speed of both computers are the same. Example: I cpu_speed 2800 [2200]		No response!
`I number_players <number>` Informs the caiaio about the number of client players. The command is compulsory. Example: I number_players 2		No response!
`I player <number> <name> <time> [<log>]` Informs the caiaio about the executable name of a client player, its maximum competition playing time in milliseconds and the name of the log file where the stderr of the client player must be written to (if omitted, the stderr will only be printed to the screen in the debug mode). The command is compulsory for all players. Example: I player 1 player1 30000 [log1.txt]		No response!

`I start <number> [<err_time>]`
Starts the client player. If the firsterror of a player is wanted, the waiting time in milliseconds must be included. The command is compulsory, the option is not.
Example: I start 1 [1000]
		If the option is included, you can expect one of the two possibilities:
		`no_firsterror`
		`firsterror <id_player>`
		Example: firsterror player1

`I referee <name> [<log>]` Starts the referee of which the executable name must be given. A string (i.e. the name of a log file) must be included if you want to inform the referee about it. The string is included as the first argument when the referee is executed. Example: I referee referee [reflog.txt]	No!	`<report_from_referee>` Example: report player1 – player2: 12 – 0
`I kill <number>` Stops a player. This command may be omitted if you are sure the client player stops itself. Example: I kill 1		No response!
`I kill_referee` Stops the referee. This command may be omitted if you are sure the referee stops itself. Example: I kill_referee		No response!
`I stop_caiaio` Stops the Caiaio. The caiaio makes sure the manager is stopped first. The command is compulsory. Example: I stop_caiaio		No response!

The protocol from the point of view of the referee

Command:	Lock?	Expected immediate response:
`<number> <message>` Sends a message to one of the client players. It is preferred to lock the referee to the caiaio first. Only the message will be sent to the player. This command must be followed immediately by a listen command if an answer is expected. Example: 1 Start	Prefer.	No primarily response!

`I listen <number>`
Makes the caiaio listen to a client player exclusively. After having communicated with one of the client players, it is wise to unlock the referee temporarily from the Caiaio.
Example: I listen 1
	Prefer.	You can expected one of the five possibilities:
		`<answer_from_player>`
		Example: d4-d5
		`<caiaio:_player_stopped>`
		`<caiaio:_player_crashed_%s>`
		Remark: %s is replaced by the type of crash
		`<caiaio:_player_timeout>`
		`<caiaio:_player_sends_"\n">`

`I request_time <number>` Requests the playing time of a client player. Example: I request_time 1		`time <actual> <estimate>` The first time is the actual playing time in milliseconds and the second is an estimate for the case the game should have been played on the competition computer. Example: time 15678 12318
`M <report>` Sends a message to the manager. The command is compulsory. Only the message will be sent to the manager. After sending the message to the manager the referee must stop itself. Example: M report player1 – player2: 12 – 0	No!	No response!

Remark: the first three commands may be used by the manager as well. This makes it in theory possible not to use the referee.

Time measurements
The caiaio makes use of the internal clock of the computer to measure the playing time of the client players. If using the C function gettimeofday[11] the actual relative time is returned in microseconds accurate. The difference in playing time measured by the caiaio and the client player is better than 2 % of the maximum playing time for games that run longer than 10 seconds. For programs that run shorter than 5 seconds inaccuracies are reported of approximately 1 second shorter in running time. The reason for this is that during the measurement of one move the caiaio misses for a non-known reason approximately 5 milliseconds.

[11] The same counts for the use of the function GetTime in Pascal.

Using Java class files with Caia[12]

With a little effort you can easily use Java class files in Caia competitions. We have written the javawrapper for that. You can find the source of *javawrapper.c* in the folder *caia_install_<namegame>/javawrapper/*.

Suppose the name of your class file is *JavaPlayer.class*. The only thing you will have to do is to rename the executable in the *bin/* folder from *javawrapper* to *JavaPlayer*.

In manager.txt you can use the program name *JavaPlayer* which refers to the executable that starts your Java player.

Using your own executable Java jar with Caia

You can also use your own Java jar in Caia competitions. We have written the jarwrapper for that.
You can find the source of *jarwrapper.c* in the folder *caia_install_<namegame>/jarwrapper/*.

Suppose the name of your class file is *JavaPlayer.class*. The only thing you will have to do is to rename the executable in the *bin/* folder from *jarwrapper* to *JavaPlayer*. The executable now will perform the command: java -jar JavaPlayer. The jar file should contain a manifest which points to the class with the main method[13].

In manager.txt you can use the program name *JavaPlayer* which refers to the executable that starts your Java jar player.

[12] In October 2007 a solution on the CodeCup forum is published on how to run two Java players against each other. Have a look in the thread “Duplicate class files” in the forum “Technical questions”.

[13] For more information: http://java.sun.com/docs/books/tutorial/deployment/jar/appman.html